Army Research Laboratory

User's Manual for the Scanning Fast Field Program (SCAFFIP) EOSAEL Version 1.0

by John M. Noble
Dave Marlin
Battlefield Environment Directorate

ARL-TR-282

January 1995

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE January 1995	3. REPORT TYPE AND DATES COVERED

4. TITLE AND SUBTITLE User's Manual for the Scanning Fast Field Program (SCAFFIP) EOSAEL Version 1.0	5. FUNDING NUMBERS
6. AUTHOR(S) John M. Noble, Dave Marlin	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army Research Laboratory Battlefield Environment Directorate ATTN: AMSRL-BE-S White Sands Missile Range, NM 88002	8. PERFORMING ORGANIZATION REPORT NUMBER ARL-TR-282
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Army Research Laboratory 2800 Powder Mill Road Adelphi, MD 20783-1145	10. SPONSORING/MONITORING AGENCY REPORT NUMBER ARL-TR-282

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Scanning Fast Field Program (SCAFFIP) is an atmospheric acoustic propagation model incorporating many of the effects on the environment on the sound field such as geometrical spreading, refraction, diffraction, molecular absorption, and complex ground impedance. SCAFFIP provides range estimations from the sensor or target for signal-noise of -10, -5, 0, 5, and 10 dB (re: 20 μPa) with azimuth for a given geometry, sound level of target at a given frequency, and meteorological profile. The meteorological profile and the geometry provides the model the ability to calculate the sound speed profile. The geometry profile is required because the angular dependence of the sound speed on the wind direction is relative to the direction of propagation. This model works over a flat-earth and a non-turbulent atmosphere. Even with these restrictions, the model performs very well for many scenarios. The model contains a friendly user interface requiring a minimum amount of information to run the model. There are also flags that can be set to obtain more detailed information.

14. SUBJECT TERMS FFP, EOSAEL, acoustic, model, atmospheric, Fast-Field Program	15. NUMBER OF PAGES 117
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT SAR

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

Contents

Figures

Tables

1. Introduction

The Scanning Fast Field Program (SCAFFIP) is based on the Fast Field Program (FFP) with the added ability to scan multiple azimuths to predict the propagation conditions about the location of a sensor. SCAFFIP makes a prediction of the acoustic propagation conditions based on spherical spreading, molecular absorption, refraction, acoustically complex ground impedance, and diffraction over benign terrain.

The FFP is a one-way solution to the acoustic-wave equation originally developed for underwater sound propagation predictions. [1,2] Raspet et al. [3] and Lee et al. [4] adapted the FFP to propagation in the atmosphere. The FFP developed by Raspet et al. used a propagation matrix formulation. If each layer in the atmosphere is viewed as an optical device, a matrix for each layer in the atmosphere can be constructed. Multiplying each matrix together results in a new matrix representing how an acoustic signal will be affected as it propagates through the atmosphere. Next, a Bessel Function Transform of the problem is taken with respect to range. After the solution is calculated, an inverse transform is performed to arrive at the final solution.

The software package that comes with SCAFFIP is an integrated set of algorithms for running the acoustic propagation model on any computer platform containing an ANSI version of a C compiler. There is an option of using the nongraphical user interface that comes with the package for extracting the propagation model and incorporating it into the user interface environment (see appendix D). Example test cases (in appendix B) show the structure of the input data files used by SCAFFIP and give a basis to work with to investigate the installation of the files. The software package is available on a variety of computer media. A list of possible media is 4-mm Digital Audio Tape (TAR or CPIO format), 0.25-in. cartridge tape (TAR or CPIO format), 8-mm Exabyte tape (TAR or CPIO format), MS-DOS 3.5-in. high- or low-density diskettes, or 0.5-in. 9-track magnetic tape (TAR or CPIO format).

2. Theory

2.1 Speed of Sound

Meteorological phenomena can have a significant effect on the received sound field. Some of the meteorological variables affecting the speed of sound in air are: pressure, temperature, vector wind speed, and humidity. To observe the effect of each meteorological variable, consider each one independently and examine the equation for the speed of sound in air. The value of c, according to Laplace's adiabatic assumption for air as an ideal gas, [5] is

$$c\ (T) = \sqrt{\frac{\gamma RT}{M}} \tag{1}$$

where

γ	=	the ratio of specific heats
R	=	the universal gas constant equal to 8314.16 J/(kg K)
M	=	the molecular weight.

The presence of water molecules alters the sound speed by lowering γ and decreasing M. The decrease in M dominates so that the overall effect of increasing humidity is an increasing sound speed. These changes can be quantified as

$$\gamma = \frac{7 + h}{5 + h} \tag{2}$$

and

$$M = 29 - 11h \tag{3}$$

where

h = the fraction of water molecules in air.

As the amount of water in the atmosphere increases, the molecular weight of a parcel of air will decrease because the molecular weight of a water molecule is less than diatomic nitrogen. This molecular weight effect will try to increase the sound speed as the fraction of water molecules in the air increases. To calculate the fraction of water molecules in air, the Goff-Gratch equation, equation (4), must be used to first calculate the partial pressure of saturated water vapor P_{sat} at temperature T.

$$\log_{10}\left(\frac{P_{sat}}{P_o}\right) = 10.79586\left[1-\left(\frac{T_{01}}{T}\right)\right] - 5.02808\log_{10}\left(\frac{T}{T_{01}}\right) + 1.50474 x 10^{-4}\left(1-10^{-8.29692[(T/T_{01})-1]}\right) + 0.42873 x 10^{-3}\left(10^{4.76955[1-(T_{01}/T)]}-1\right) - 2.2195983 \tag{4}$$

where

T_{01} = 273.16 K
p_o = 1 atm or the reference pressure.

After the value for P_{sat} is determined, the fraction of water molecules in air can be calculated using the following relationship:

$$h = \frac{10^{-2}(RH)P_{sat}}{P} \tag{5}$$

where

RH = the relative humidity in percent
P = the pressure in atmospheres.

The magnitude of the dependence of the sound speed on humidity is not obvious. To understand the degree of the effect of humidity on sound speed, consider a particular case. At 20 °C, the difference in sound speed between 0 and 100 percent humidity is 2 m/s. A fluctuation in the humidity of this amount is very unlikely. If the variation in humidity is reduced to a change of 50 to 100 percent, the change in the sound speed is only 1 m/s. Therefore, the variation of sound speed caused by changes in humidity should always be much less than 1 m/s. Generally, humidity fluctuations can be ignored.

The effect of the wind speed on the speed of sound is a vector relation. The effective sound speed is calculated using

$$c_{eff} = c(T) + u \cdot \cos(\theta_w - \pi - \theta_R) \tag{6}$$

where

c(T)	=	the speed of sound in the absence of wind at temperature T
u	=	the magnitude of the horizontal wind speed
θ_R	=	the bearing of the receiver from the source
θ_w	=	the direction from which the wind blows
$\theta_w - \pi$	=	the direction the wind is blowing (figure 1).

All directions are relative to north.

The sound speed will also vary with height because the sound speed is a function of temperature and vector wind speed. This variation will cause the acoustic wave to be refracted as it propagates through the atmosphere. The degree of refraction the acoustic wave undergoes is related to the sound speed gradient present in the atmosphere. If the sound speed increases with height, the acoustic wave will be refracted downwards. If the sound speed decreases with height, the acoustic wave will be refracted upwards.

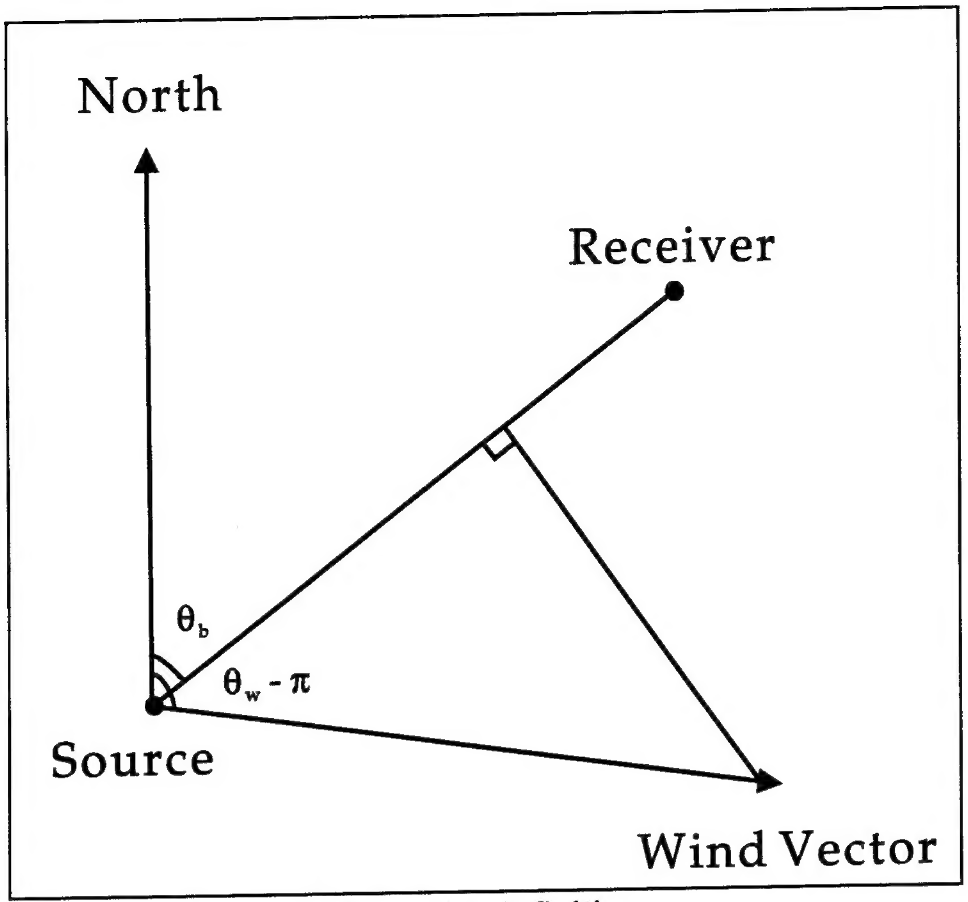

Figure 1. Diagram of geometry definition.

2.2 FFP

The propagation of sound from a point source located at the origin is given by the classical wave equation

$$\nabla^2 p - \frac{1}{c^2}\frac{\partial^2 p}{\partial t^2} = -4\pi\,\delta(x,y,z) \tag{7}$$

where δ represents a delta function source of unit strength. For simple harmonic motion, equation (7) becomes the Helmholtz equation

$$\nabla^2 p + k^2 p = -4\pi\,\delta(x,y,z) \tag{8}$$

where

k = is the wavenumber, ω/c, in which c = the sound speed, ω = the angular frequency.

For the FFP, k and c are restricted to vary only in the z-direction.

Transforming equation (8) into cylindrical coordinates and assuming azimuthal symmetry, the Helmholtz equation becomes

$$\frac{\partial^2 p}{\partial r^2} + \frac{1}{r}\frac{\partial p}{\partial r} + \rho\frac{\partial}{\partial z}\left(\frac{1}{\rho}\frac{\partial p}{\partial z}\right) + k^2 p = -\frac{2}{r}\,\delta(r)\,\delta(z-z_s) \tag{9}$$

where the source is located at r = 0 and z = z_s and ρ is the density of the medium.

The atmosphere is viewed as a series of constant sound speed layers for the FFP (as shown in figure 2). The layers in the atmosphere are bounded on top and bottom by complex impedance surfaces. The top boundary is typically modeled as an infinite half-space with constant parameters. At the bottom boundary, the atmospheric layer adjoins a partially absorbing surface that can be represented by the complex acoustical impedance of the ground.

To reduce the dimensionality of equation (9), a zero-order Hankel transform is applied with respect to the range variable r. This gives the transform pair:

$$\begin{aligned}\check{p}(\kappa,z) &= \int_0^\infty p(r,z)\,J_o(\kappa r)\,r\,dr \\ p(r,z) &= \int_0^\infty \check{p}(\kappa,z)\,J_o(\kappa r)\,\kappa\,d\kappa.\end{aligned} \tag{10}$$

Applying the first transform to equation (9) results in

$$\frac{d^2\check{p}}{dz^2} + [k^2(z) - \kappa^2]\,\check{p} = -2\delta(z-z_s). \tag{11}$$

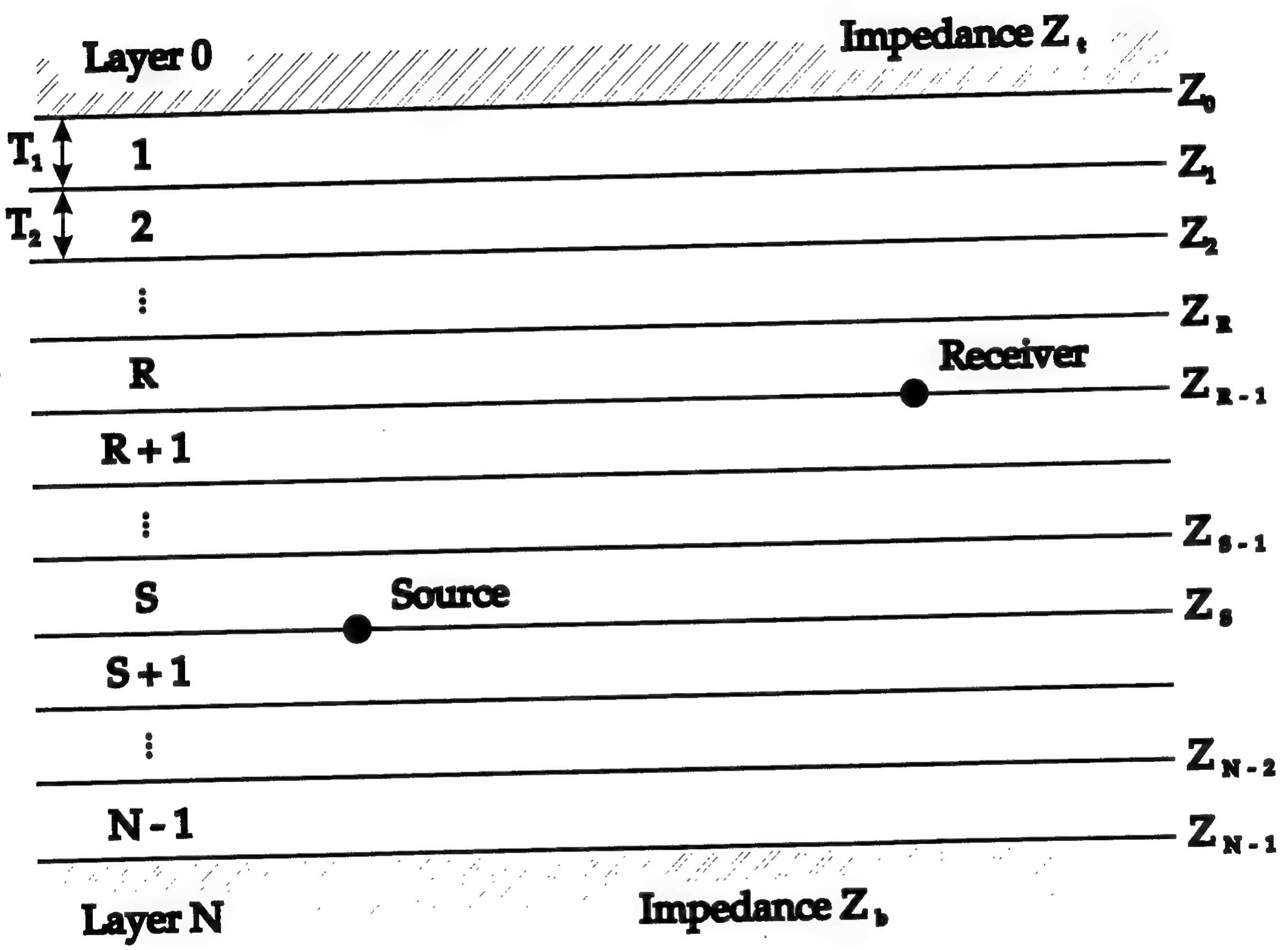

Figure 2. Layering of the atmosphere by the FFP.

This equation can be decomposed into

$$\check{u}_z = \frac{i}{\omega \rho_o} \frac{d\check{p}}{dz} \tag{12}$$

and

$$\frac{d\check{u}_z}{dz} = \frac{i}{\omega \rho_o} \frac{d^2\check{p}}{dz^2} = -\frac{i}{\omega \rho_o} [\ k^2(z) - \kappa^2\ ]\ \check{p} + \frac{2}{i\omega \rho_o} \delta(z - z_s) \tag{13}$$

where

ρ_o = the mean air density

$\check{u}_z$ = the transformed particle velocity in the z-direction.

The delta function is the source term producing a discontinuity in $\check{u}_z$ at a height z_s. To solve the resulting equations, Lee et al. [4] used an analogy to a transmission line, which results from the form of the transformed equation. The transformed equations have a very similar form to the telegrapher's equations of electrical transmission line theory. From the analogous telegrapher's equations,

$$\frac{dV}{dz} = -Z\,I(z) \tag{14}$$

and

$$d\,I(z)\,dz = -Y(z)\,V(z) + I_o\,\delta(z - z_s) \tag{15}$$

there is a similar form of the equations if the shunt admittance Y(z) is made a function of z and the series impedance Z is made a constant with a current source at $z = z_s$. The equivalent transmission line configuration to figure 2 is shown in figure 3.

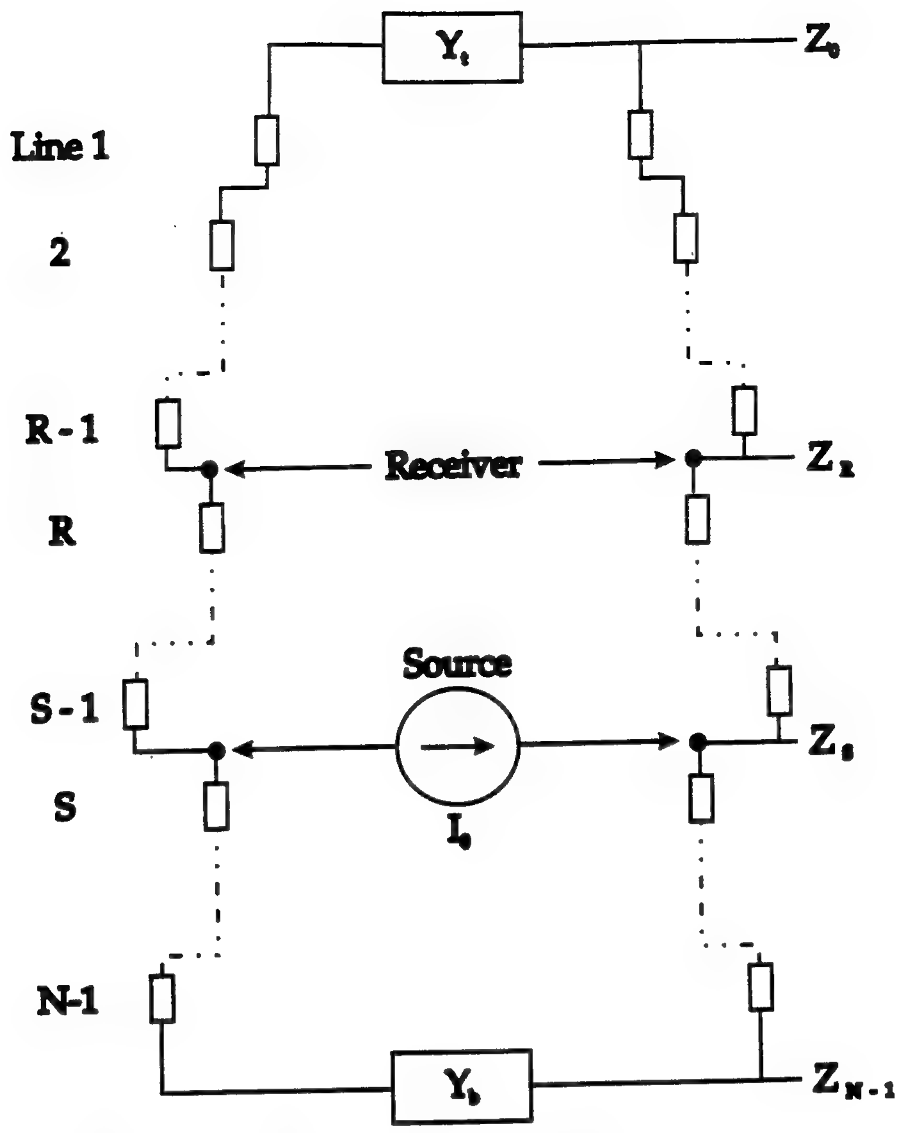

Figure 3. Transmission line analog used by the FFP.

Using the Lee et al. analogy, the acoustic problem can be arranged so that a solution can be calculated. The analogy is made by representing each layer in the atmosphere by an element in a transmission line with a certain characteristic admittance and attenuation constant. The admittance is defined as one over the impedance or one over the sum of the resistance plus the reactance of the electrical element. The admittance of the element causes the voltage running through it to be attenuated and a shift in the phase of the signal. Viewing the analogy from the perspective of the acoustic wave problem, as the acoustic wave is propagated through the atmosphere, it undergoes losses and phase shifting caused by refraction and spreading of the acoustic wave. The equivalence between the transmission line and the acoustic wave problem is illustrated in table 1. This equivalence can be carried out for each layer of the atmosphere; thus, constructing a transmission line. The top and bottom boundaries in the atmosphere become loading admittance elements on each end of the transmission line. The problem now has been converted from determining the voltage in the transmission line to a point in the line. This is a well-known process in electrical engineering.

Table 1. Transmission line analog relationships to acoustic fields

	Acoustic Wave	Transmission Line
Fields	Pressure $p(k,z)$	Voltage $V(z)$
	Velocity $\dot{\omega}(k,z)$	Current $I(z)$
Attenuation Constant	$\gamma = \sqrt{k^2 - (\omega/\alpha)^2}$	$\gamma = \sqrt{ZI}$
Characteristic Admittance	$Y_C = -\gamma/\omega\rho$	$Y_C = \sqrt{Y/Z}$

The zero-order Bessel function in equation (10) can be expanded using Hankel functions:

$$J_o(\kappa r) = \frac{1}{2}\,[\,H_o^{(1)}(\kappa r) + H_o^{(2)}(\kappa r)] \tag{16}$$

The Hankel functions can be represented as an incoming and outgoing acoustic wave. The FFP is designed to model radially outgoing acoustic waves. This allows for the first Hankel function to be suppressed with the additional argument that the incoming acoustic wave will not contribute significantly to the final result. The asymptotic expansion of the second Hankel function for large arguments is as follows:

$$H_o^{(2)}(\kappa r) \approx \sqrt{\frac{2}{\pi\kappa}}\;\frac{e^{-i(\kappa r - \pi/4)}}{\sqrt{r}} \qquad \kappa r \gg 1 \tag{17}$$

making the problem easier to handle. The important contributions from the integrand of the inverse transform, equation (10), comes from the area where $\kappa \sim k_o$. Substituting equation (17) into the inverse transform of equation (10) and taking the far-field approximation, the acoustic pressure equation can be written as

$$p\,(r, z_r) \approx \frac{(1+i)}{\sqrt{2\pi r}} \int_0^\infty \check{p}\,(\kappa, z_r)\, e^{-i\kappa r}\sqrt{\kappa}\; d\kappa . \tag{18}$$

To perform the calculation on a computer, the continuous integral must be replaced by a numerical integral over discrete values of κ. Applying this to equation (18) yields

$$p(r,z_r) = \frac{(1+i)}{\sqrt{2\pi r}}\;\Delta\kappa \sum_{n=0}^{N-1} \check{p}(\kappa_n)\sqrt{\kappa_n}\;\; e^{-i\,(2\pi n m/N)} \tag{19}$$

where

$$\Delta\kappa = \frac{\kappa_{\max}}{N-1} \qquad \kappa_n = n\Delta\kappa$$
$$m = \frac{r}{\Delta r} \qquad \Delta r = \frac{2\pi}{N\Delta\kappa}. \tag{20}$$

The term $\kappa_{\max}$ comes from the property of the integrand of equation (10) and only has significant contributions in a finite range of κ, allowing the summation to be terminated at a finite number of terms.

A problem in the derivation is one of the complex numerical integrations. The function being integrated contains branch points and poles on the real axis. Because of the nature of branch points and poles, the integration being performed must not include any of these points for a correct solution. To avoid these problems, the current FFP uses what is called extra loss in the calculations. The mathematical result of using this extra loss is to move the numerical integration off the real axis. The effect of the extra loss is removed from the solution in an approximate manner by multiplying the computed pressure by the term exp(αr), where α is the extra loss attenuation constant in Np/m. The proper choice of the artificial attenuation is essential if meaningful results are to be obtained from the code.

Another problem with the numerical integration is the number of points N used in the summation of equation (19). A lower bound on the number of points required in the summation is

$$N_{\min} = \frac{\kappa_{\max} r}{\pi}. \text{ [6]} \tag{21}$$

However in most cases, this lower bound is too large to perform one single Fast Fourier Transform (FFT). The problem is if there are enough points to sufficiently sample the wavenumber space in the numerical integration. If equation (19) is rewritten in the form

$$S = \sum_{n=0}^{N-1} G_n e^{-i(2\pi nm/N)}, \tag{22}$$

this summation [6] can be rewritten so that the single summation is rearranged to a double summation of the form

$$S = \sum_{b=1}^{p} e^{-i[2\pi(b-1)m/p]} \sum_{n=0}^{N'-1} G_{n+(b-1)N'} e^{-i[2\pi nm/(pN')]} \tag{23}$$

where

N´	= N/p
p	= an integer larger than 1.

p is the number of panels that the original FFT has been divided into. Each panel contains N´ points. This technique allows the FFT to be performed to calculate the acoustic pressure with range at the height of the receiver.

2.3 Absorption of Sound in the Air

Losses in the medium are basically caused by viscosity, heat conduction, and molecular exchanges of energy. In the nineteenth century, only the mechanisms of viscosity and heat conduction were suspected of causing dissipation of sound; therefore, they are presently referred to as classical absorption.

In classical absorption, if the effect of absorption is represented by a factor $e^{-\alpha r}$ where r is the distance of propagation, then the attenuation coefficient α_{cl} caused by viscosity and heat conduction is given by equation (17) from Physical Acoustics XVII [7]:

$$\alpha_{cl} = 5.578 \ x \ 10^{-9} \frac{T/T_o}{T+110.4} \frac{f^2}{P/P_o}. \tag{24}$$

The units of α_{cl} in equation (24) are nepers/meter, where

P_o	=	the reference pressure of 1.01325×10^5 Npm^{-2} (1 atm)
P	=	pressure in Npm^{-2}
T_o	=	the reference temperature of 293.15 K
T	=	temperature in K
f	=	frequency (Hz)

In molecular absorption, energy exchanges at the molecular level include rotational and vibrational modes. Analysis of the rotational mode shows that the representative attenuation coefficient is proportional to α_{cl}, the classical attenuation coefficient:

$$\frac{\alpha_{rot}}{\alpha_{cl}} = 4.16\, e^{-16.8\,T^{-1/3}} \qquad (25)$$

when $293 \text{ K} < T < 690 \text{ K}$.

For frequencies below 10 MHz, it has been demonstrated that energy losses caused by classical and molecular absorption are additive.

$$\alpha_{cr} = \alpha_{cl} + \alpha_{rot}. \qquad (26)$$

A simplified empirical form of the equation can be written as

$$\alpha_{cr} = 1.83 \ x \ 10^{-11} \frac{\sqrt{T/T_o}\, f^2}{P/P_o}, \qquad (27)$$

which is correct within 2 percent for $213 \text{ K} < T < 373 \text{ K}$.

The vibrational mode of absorption should also be considered. Because the atmosphere is composed mostly of nitrogen and oxygen, each will contribute an attenuation coefficient, where j stands for either oxygen or nitrogen. The symbols are defined as follows:

$$\alpha_{vib,j} = \frac{4pX_j}{35c}\left(\frac{q_j}{T}\right)^2 \frac{e^{-q_j/(Tf^2)}}{f_{r,j} + (f^2/f_{r,j})} \tag{28}$$

X_j	=	the mole fraction of air component considered (0.20948 for oxygen and 0.78084 for nitrogen)
q_j	=	the characteristic vibrational temperature (2239.1 K for oxygen and 3352.0 K for nitrogen)
c	=	the speed of sound at temperature T (m/s).

The $f_{r,j}$ are the individual relaxation frequencies for oxygen and nitrogen. The computation of these frequencies depends on the relative humidity and atmospheric pressure. The relaxation frequencies are given by:

$$f_{r,O} = \frac{P}{P_o}\left(24 + 4.04 \, x \, 10^4 h \frac{0.02 + h}{0.391 + h}\right)$$

$$f_{r,N} = \frac{P}{P_o}\sqrt{\frac{T_o}{T}}\left(9 + 280\, h e^{-4.170\,(\,T_o/T\,)^{1/3} - 1}\right). \tag{29}$$

The total attenuation coefficient is then the sum of α_{cr} and $\alpha_{vib,j}$. Figure 4 is a log-log plot [6] of total attenuation coefficient for T = 20 °C and Rh = 20 percent. Figure 4 shows the attenuation caused by classical absorption, vibrational relaxation of nitrogen and oxygen, and the total attenuation coefficient caused by the sum of the three attenuation mechanisms.

The attenuation coefficient α is proportional to the square of the frequency. As the frequency doubles, the attenuation will quadruple. The attenuation of the sound wave caused by molecular and vibrational absorption is very important for frequencies over 250 Hz. For frequencies below 250 Hz, this attenuation does not contribute much to the total attenuation of the sound wave.

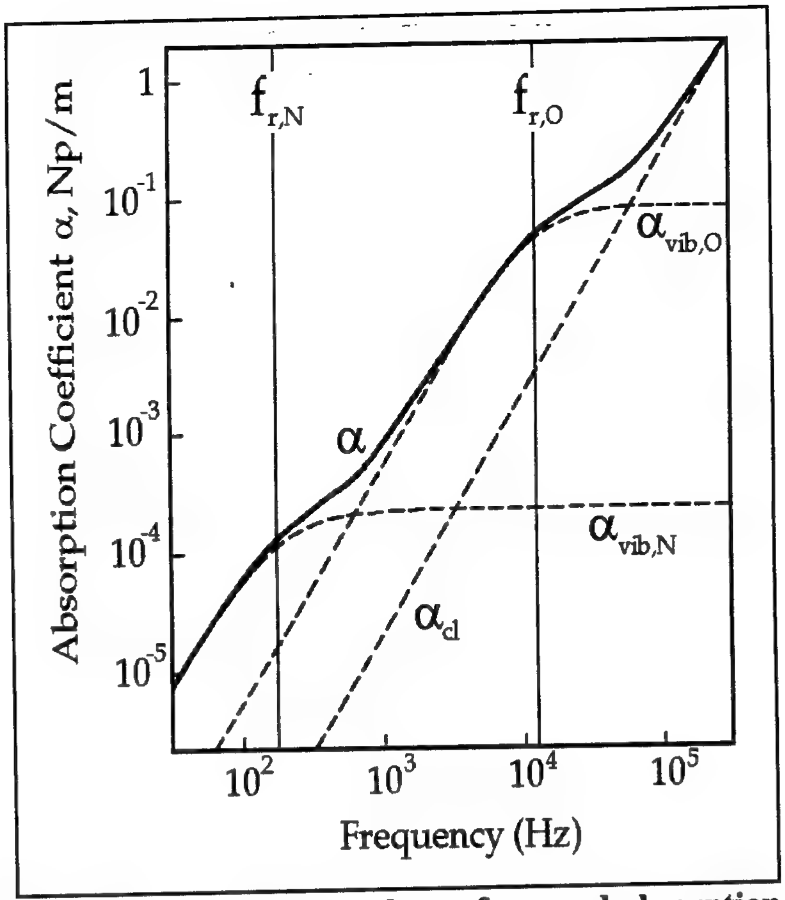

Figure 4. **Log-log plot of sound-absorption coefficient versus frequency for sound in air at 20 °C at 1 atm pressure and with a water vapor fraction h of 4.676 x 10^{-3} (Rh = 20%).**

2.4 Complex Ground Impedance

There are several models available for calculating the complex ground impedance. The impedance model used in SCAFFIP is the Four Parameter Model of Attenborough. [8] In this impedance model, the complex normalized characteristic impedance of the ground is calculated using

$$Z_c \cong \frac{\left[\dfrac{4q^2}{3\Omega} + i\dfrac{S_f^2\sigma}{\omega\rho_o}\right]}{k_b} \tag{30}$$

where

q^2	=	$\Omega^{-n'}$
S_f	=	the pore shape factor ratio
Ω	=	the porosity of the ground
σ	=	the flow resistivity of the ground (mks) rayls
ω	=	the angular frequency
ρ_o	=	the density of air (1.2 kg/m^3)
k_b	=	the normalized wave number.

The normalized wave number k_b is computed from

$$k_b \cong \sqrt{\gamma\,\Omega}\left[\left(\frac{4}{3} - \frac{\gamma - 1}{\gamma}N_{pr}\right)\frac{q^2}{\Omega} + i\frac{S_f^2\,\sigma}{\omega\,\rho_o}\right]^{1/2} \tag{31}$$

where

γ	= the ratio of specific heats (equation (2))
N_{pr}	= the Prandtl number (0.724).

The parameters S_f, Ω, ρ, and n′ are normally varied until agreement is reached between impedance measurements and the impedance model is achieved. However, this method of determining the four parameters cannot always be completed if time or resources are lacking. Table 2 provides rough estimates of the values for the four parameters, given some general descriptions of a variety of ground surfaces, to aid the user when the four parameters are unknown.

Some ground surfaces have a layered structure resulting from the gradual deposition of material over a soil base (a layer of snow over frozen ground, decomposition of organic material over clay or sandy soil, or a well plowed pasture over clay or harder packed soil.) An effective impedance Z(d) can be calculated for a semi-infinite layer of impedance Z_2 covered by a layer, depth d, of another material of impedance Z_1. The effective impedance is given by

$$Z(d) = \left[\frac{Z_2 - iZ_1 \tan(k_b d)}{Z_1 - iZ_2 \tan(k_b d)} \right] Z_1 \tag{32}$$

where

k_b = the bulk propagation constant in the top layer of the ground.

The parameters Z_1, Z_2, and k_b are calculated using equations (30) and (31) from Attenborough's Four Parameter impedance model.

Table 2. Estimation of Ω and σ for various ground surfaces given $n' = 0.750$ and $S_f = 0.875$

Description of Surface	Ω	σ(cgs)
Dry snow, new fallen 0.1 m over about 0.4 m old snow	0.850	23
Sugar snow	0.825	48
In forest, pine, or hemlock	0.825	48
Grass: rough pasture, airport, public buildings, etc.	0.675	330
Roadside dirt, ill-defined, small rocks up to 0.1 m mesh	0.575	960
Sandy silt, hard-packed by vehicles	0.475	3470
"Clean" limestone chips, thick layer (0.01 to 0.025 m mesh)	0.425	6470
Old dirt roadway, fine stones (0.5 m mesh) interstices filled	0.400	7500
Earth, exposed and rain-packed	0.350	17100
Quarry dust, fine, very hard-packed	0.300	41700
Asphalt, sealed by dust and light use	0.250	120000

3. Operations

3.1 Purpose of User Interface

The user interface is designed to run the model with numerous scenarios and a minimum of modifications to the input files between each run. The user interface also allows the model to be run interactively or in a batch job format. This user interface will be standard for all of the benign terrain acoustic propagation models released by the U.S. Army Research Laboratory Battlefield Environment Directorate (ARL/BED). This allows for the same data files to be used for future benign terrain acoustic propagation models obtained from ARL/BED. If other data input formats are required, ARL/BED can assist with the modifications to the current user interface. Under most conditions, ARL/BED cannot guarantee compatibility with future upgrades and acoustic propagation models after the original user interface has been modified.

3.2 Format of Input Data Files

There are two categories of input cards required by the EOSAEL version of SCAFFIP. Example input files are included with the discussion of test cases in appendix B. The two input files are called scaffip.i and met.i. The syntax of each of the two card categories is given below:

3.2.1 Scaffip.i

The scaffip.i file contains the information relating to the geometry of the problem, frequency, parameters relating to ground impedance, print location, and optional output (ground impedance, speed of sound profile, and wavenumber spectrum). An integer code allows for input of additional parameters normally fixed within the program and output of additional files.

The file format for scaffip.i follows:

Code	- input/output code options - integer
Print	- location of output: file (1), screen (2), or both (3) - integer
Zs	- source height in meters - real
Zr	- detector height in meters - real
Range	- horizontal distance between source and detector - real
Azimuth1	- starting azimuth of scan in degrees - real
Azimuth2	- finishing azimuth of scan in degrees - real
Dazimuth	- azimuthal resolution of scan in degrees - real
Frequency	- frequency of interest in hertz - real
Src_Level	- level of source at frequency in decibel - real
Back_Level	- level of background noise at frequency in decibel - real
N_Ground	- number of ground layers (1 or 2) - integer
Sigma1	- flow resistivity of top ground layer in cgs rayls - real
Omega1	- porosity of the top ground layer - real
Pn1	- grain shape factor of the top ground layer - real
Sf1	- pore shape factor of the top ground layer - real
D	- depth of the surface layer in meters - real
Sigma2	- flow resistivity of bottom ground layer in cgs rayls - real
Omega2	- porosity of the bottom ground layer - real
Pn2	- grain shape factor of the bottom ground layer - real
Sf2	- pore shape factor of the bottom ground layer - real

The input code parameters for scaffip.i follow:

Code:

0	Normal operation
1	Output speed of sound profile (sound.o)
2	Use radians for wind direction
4	Source centered calculation
8	Input extra loss (extra.los)
16	Input number of points and panels for performing FFT (trans.frm)
32	Output panel values (npan.e)
64	Output ground impedance (imped.e)
128	Output wave number spectrum (wavnum.e)

The value for code can be one of the numbers listed above or a sum of any two or more of those numbers. If normal operation of the model is desired, use a zero in the debug file. If the speed of sound profile and wave number spectrum output was desired, the value in the debug file would be 129 = 1 + 128.

The ground parameters for scaffip.i are discussed here.

Not all the entries are required. If only one ground layer is used, set Nground = 1, then D, Sigma2, Om2, Pn2, Sf2 may be omitted from the file. If Nground = 1, then values for D, Sigma2, Om2, Pn2, Sf2 may be present in the file; however, the program will not read any of these values. Recommended values for these parameters are given in table 1 if measurements are not available.

3.2.2 *Met.i*

The met.i file contains the atmospheric profile. The first line of the file is an integer. The rest of the file contains real floating point numbers.

The file format for met.i follows:

```
Nint
 Z(1)     T(1)      Rh(1)      P(1)      Wvel(1)     Wdir(1)
 Z(2)     T(2)      Rh(2)      P(2)      Wvel(2)     Wdir(2)
  ⋮        ⋮         ⋮          ⋮          ⋮           ⋮
Z(Nint)  T(Nint)   Rh(Nint)   P(Nint)   Wvel(Nint)  Wdir(Nint)
```

where

Nint	- number of interfaces (There is no limit; however, many interfaces will slow the program down. Try and limit the number of interfaces to 60.)
Z(i)	- height of the ith interface in meters, also Z(1) = 0 and $Z(i) < Z(i+1)$ always.
T(i)	- temperature in Kelvin at height Z(i).

Rh(i)	-	relative humidity in percent at height Z(i).
P(i)	-	pressure in atmosphere at height Z(i).
Wvel(i)	-	wind speed in meters per second at height Z(i).
Wdir(i)	-	direction, in degrees, from which the wind is blowing relative to north at height Z(i).

3.3 How To Run SCAFFIP

SCAFFIP is relatively easy to run. There is a C-Shell script included with the source code to compile the executable on a Unix workstation. To run the model, type **sffpeo** at the command line. The program will read the two files scaffip.i and met.i, compute the environmental effects on the acoustic signal, calculate the ranges, and print the results in scaffip.o if requested. Appendix B contains four example cases. The example cases cover the benchmark cases soon to be published in the *Journal of the Acoustical Society of America*.

3.4 Format of the Output

The output from the model is saved to a file named scaffip.o if the file option is set. The format of the output file follows:

```
                    Scanning Fast Field Program (SCAFFIP)
                                EOSAEL Version
-------------------------------------------------------------------------------
SOURCE HEIGHT     =   150.0 METERS          FREQUENCY          =   20.0 HZ
DETECTOR HEIGHT   =   2.0 METERS            DETECTOR CENTERED
SOURCE LEVEL      =   120.0 DECIBELS        BACKGROUND LEVEL   =   60.0 DECIBELS
MAXIMUM RANGE     =   5000.0 METERS
-------------------------------------------------------------------------------
```

DETECTOR RANGE (METERS) OVER BENIGN TERRAIN

	SIGNAL - NOISE (dB)				
AZIMUTH (DEG)	-10	-5	0	5	10
Azimuth1	R1(-10)	R1(-5)	R1(0)	R1(5)	R1(10)
Azimuth2	R2(-10)	R2(-5)	R2(0)	R2(5)	R2(10)
⋮	⋮	⋮	⋮	⋮	⋮
Azimuth#	R#(-10)	R#(-5)	R#(0)	R#(5)	R#(10)

where Azimuth# is the azimuth in degrees and R#(S-N) is the range in meters when the signal reaches the signal level relative to the background noise level. If the program does not find a range for the specific signal - noise level and azimuth, a value of -999.0 will be displayed.

References

1. DiNapoli, F. R., Naval Underwater Systems Center, "A Fast Field Program for Multilayered Media", Technical Report 4103, 1971.

2. Kutschale, H. W., "The Integral Solution of the Sound Field in a Multilayered Liquid-solid Half-space with Numerical Computations for Low-frequency Propagation in the Arctic Ocean," Report No. 1 (CU1-1-70, ONR Contract N00014-67-A-0108-0016), Lamont-Doherty Geological Observatory, Columbia University, Palisades, NY, 1970.

3. Raspet, R., S. W. Lee, E. Kuester, D. C. Chang, W. F. Richards, R. Gilbert, and N. Bong, "Fast-field Program for a Layered Medium Bounded by Complex Impedance Surfaces," *J. Acoust. Soc. Am.* **77**:345-352, 1985.

4. Lee, S. W., N. Bong, W. F. Richards, and R. Raspet, "Impedance Formulation of the Fast Field Program for Acoustic Wave Propagation in the Atmosphere," *J. Acoust. Soc. Am.* **79**:628-634, 1986.

5. Pierce, Allen D., *Acoustics: An Introduction to Its Physical Principles and Applications*, McGraw-Hill, New York, 1981.

6. Franke, S. J. and G. W. Swenson, Jr., "A Brief Tutorial on the Fast Field Program (FFP) as Applied to Sound Propagation in the Air," *Applied Acoustics* **27**:203-215, 1989.

7. *Physical Acoustics XVIII*, edited by Mason and Thurston, Academic Press, New York, 1984, pp. 145-232.

8. Attenborough, K., "Acoustical Impedance Models For Outdoor Surfaces," *J. Sound Vib* **99**:521-544, 1985.

Acronyms and Abbreviations

ARL	Army Research Laboratory
BED	Battlefield Environment Directorate
CPIO	Copy File Archives In and Out
FFP	Fast Field Program
FFT	Fast Fourier Transform
SCAFFIP	Scanning Fast Field Program
TAR	Tape Archive

Appendix A
Source Code For SCAFFIP
EOSAEL Version 1.0

ADMIT.C

```
/*
 *  FINDS ADMITTANCE OF ONE BRANCH OF THE TRANSMISSION LINE
 */

#include <stdio.h>
#include "complex.h"
#include "ffp.h"

void admit(admittance, gp, last, layer, voltage)
complex *admittance, *voltage;
int last;
struct gp *gp;
struct layer *layer;
{
    complex phase, ref_1, ref_2, unit, z, z1, z2;
    int i2l, index, step, toggle = 0;

/*
 *  direction to source from last (+1 == up, -1 == down)
 */
    step = gp->index_source >= last ? 1 : -1;
/*
 *  from interface to section beyond interface (+1 == up, 0 == down)
 */
    i2l = (step + 1) / 2;
/*
 *  admittance of last significant line
 */
    Re(*admittance) = Re(layer[last].y0);
    Im(*admittance) = Im(layer[last].y0);
/*
 *  step through network to section in front of source
 */
    Re(unit) = 1;
    Im(unit) = 0;
    for (index = last + step; index <= gp->index_source && step == 1
       || index > gp->index_source && step == -1; index += step)
    {
/*
 *      set toggle and voltage if beyond the receiver
 */
        if (index == gp->index_detector + i2l)
        {
            toggle = 1;
            Re(*voltage) = 1;
            Im(*voltage) = 0;
        }
/*
 *      electrical phase factor (gamma == j * kz)
 */
        Re(z) = 0;
        Im(z) = -layer[index].thickness;
        cmul(z, z, layer[index].kz);
        cexp(phase, z);
/*
 *      reflection coefficient at the end of section
 */
        csub(z1, layer[index].y0, *admittance);
        cadd(z2, layer[index].y0, *admittance);
        cdiv(ref_1, z1, z2);
/*
 *      reflection coefficient at the beginning of section
 */
        cmul(z, phase, phase);
        cmul(ref_2, ref_1, z);
/*
 *      total voltage at the beginning of section
 */
        if (toggle)
        {
            cadd(z1, unit, ref_2);
            cmul(z1, *voltage, z1);
            cadd(z2, unit, ref_1);
            cmul(z2, phase, z2);
            cdiv(*voltage, z1, z2);
        }
/*
 *      admittance at the beginning of section
 */
        csub(z1, unit, ref_2);
        cmul(z1, layer[index].y0, z1);
        cadd(z2, unit, ref_2);
        cdiv(*admittance, z1, z2);
    }
/*
 *  admittance at the source
 */
}
```

AIR.C

```
/*
 *  AIR ABSORPTION ROUTINE
 *
 *  (1)  ANSI STANDARD S1.26-198X
 *  (2)  H. E. Bass et al,
 *       Absorption of Sound by the Atmosphere,
 *       in "Physical Acoustics", 1984.
 */

#include "complex.h"

/*
 *  reference ambient temperature
 *  293.16 degrees Kelvin == 20 degrees Celsius
 */
#define T20 293.15

double air_absorption (frequency, kelvin, pressure, relative_humidity)
double frequency, kelvin, pressure, relative_humidity;
{
    double alpha, humidity, nrf, orf, saturation, square, temp;

    temp = T20 / kelvin;
/*
 *  saturation pressure
 *  reference 2, page 169, equation 72
 */
    saturation = pow(10.0, 8.422 - 10.05916 * temp + 5.023 * log10(temp));
/*
 *  percent mole fraction of water vapor
 *  reference 1, page 19, equation D10
 */
    humidity = relative_humidity * saturation / pressure;
/*
 *  oxygen relaxation frequency
 *  reference 1, page 7, equation 9
 */
    orf = pressure * (24 + 40400 * humidity
        * (humidity + 0.02) / (humidity + 0.391));
/*
 *  nitrogen relaxation frequency
 *  reference 1, page 7, equation 10
 */
    nrf = pressure * sqrt(temp)
        * (9 + 280 * humidity * exp(4.17 * (1 - pow(temp, 0.3333))));
/*
 *  absorption coefficient alpha (nepers/m)
 *  reference 1, page 7, equation 8
 */
    square = frequency * frequency;
    alpha = 0.01275 * exp(-2239.1 / kelvin) / (orf + square / orf);
    alpha += 0.1068 * exp(-3352.0 / kelvin) / (nrf + square / nrf);
    alpha = square
        * (1.84e-11 / sqrt(temp) / pressure + alpha * pow(temp, 2.5));
    return alpha;
}
```

CLIP.C

```
/*
 *  SIMPLIFY TRANSMISSION LINE BY CLIPPING TECHNIQUE OF Lee, et al
 */

#include <stdio.h>
#include "complex.h"
#include "ffp.h"

/*
 *  exp(kzmax) == 1e8
 */
#define kzmax   18.420680743952365472l4

void parameters();

void clip(gp, k, last, layer, load, n_layers, omega)
double k, omega;
int *last, load, n_layers;
struct gp *gp;
struct layer *layer;
{
    double decay = 0;
    int i2l, index, step, toggle;

    *last = -1;
/*
 *  set toggle if detector is not between load and source
 */
    toggle = gp->index_detector > gp->index_source
        ? gp->index_source > load : gp->index_source <= load;
/*
 *  direction to load from source (+1 == up, -1 == down)
 */
    step = gp->index_source <= load ? 1 : -1;
/*
 *  conversion from interface to opposite layer (+1 == up, 0 == down)
 */
    i2l = (step + 1) / 2;
/*
 *  step from source to layer nearest load
 */
    for (index = gp->index_source + i2l; index < load && step == 1
        || index > load && step == -1; index += step)
    {
```

```
/*
 *       find admittance and wave number for section
 */
         parameters(gp, k, index, layer, n_layers, omega);
/*
 *       sum the exponential decay coefficients
 */
         decay -= layer[index].thickness * Im(layer[index].kz);
/*
 *       check if beyond the detector
 */
         if (toggle)
         {
/*
 *           replace with infinite section if decay too large
 */
             if (decay >= kzmax / 2)
             {
                 *last = index;
                 return;
             }
         }
         else
         {
/*
 *           v == 0 if decay too large
 */
             if (decay >= kzmax)
                 return;
/*
 *           set toggle if detector is reached
 */
             if (index + step == gp->index_detector + i2l)
                 toggle = 1;
         }
     }
/*
 *   find parameters for the load layer
 */
     parameters(gp, k, load, layer, n_layers, omega);
/*
 *   last significant layer in the network
 */
     *last = load;
}
```

CUTOFF.C

```
#include <stddef.h>
#include <stdio.h>
#include "complex.h"
#include "ffp.h"

#define PANEL_VALUES    dflags[5]

extern int errno;

void cutoff(delta_k, delta_r, frequency, gp, k_max, k_width, layer, n_panels,
    omega, points, range, dflags)
double *delta_k, *delta_r, frequency, *k_max, *k_width, omega, range;
int *n_panels, points, *dflags;
struct gp *gp;
struct layer *layer;
{
    double c_min, difference = 0, gamma;
    int index, max, min, nyquist;
    FILE *fp_pv;

/*
 *  highest and lowest of layers between detector and source
 */
    max = gp->index_detector > gp->index_source
        ? gp->index_detector : gp->index_source;
    min = gp->index_detector + gp->index_source - max + 1;
/*
 *  minimum speed from source to detector
 */
    c_min = layer[min].c;
    for (index = min; index <= max; index++)
    {
        if (c_min > layer[index].c)
            c_min = layer[index].c;
/*
 *      total altitude difference between detector and source
 */
        difference += layer[index].thickness;
    }
/*
 *  find upper cutoff wave number kmax based on an empirical relationship
 */
```

```
    gamma = 0.0075 * frequency + (2.5 - 6.25e-4 * frequency) / difference;
    *k_max = sqrt(gamma * gamma + (omega / c_min) * (omega / c_min));
/*
 *  determine the number of panels
 */
    nyquist = *k_max * range / points / PI + 1;
    if (*n_panels == 0)
        *n_panels = nyquist;
    if (*n_panels < 0)
        *n_panels *= -nyquist;
/*
 *  index of the point nearest range based on Nyquist criteria
 */
    index = *k_max * range / *n_panels / TWO_PI + 1.5;
/*
 *  adjust range separations so that the index point is at range
 */
    *delta_r = range / (index - 1);
/*
 *  adjust upper cutoff wave number according to adjusted delta_r
 */
    *k_max = TWO_PI * *n_panels / *delta_r;
/*
 *  wave number band width per panel
 */
    *k_width = *k_max / *n_panels;
    *delta_k = *k_width / points;
/*
 *     Write Out Panel Values
 */
    if (PANEL_VALUES)
    {
        if ((fp_pv = fopen("npan.e","a")) == NULL)
        {
            fprintf(stderr,"error number %d\n",errno);
            perror("npan.e");
            exit(1);
        }
        fprintf(fp_pv,"Kmax = %10.4lf\tNyq = %5d\tNpan = %5d\n"
            ,*k_max,nyquist,*n_panels);
        fprintf(fp_pv,"Kwidth = %10.4lf\tDelR = %10.4lf\tDelK = %10.4lf\n"
            ,*k_width,*delta_r,*delta_k);
        fclose(fp_pv);
    }
}
```

DRANGE.C

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include "complex.h"
#include "ffp.h"

/*
 *  print a line of 80 dashes
 */
#define dash\
    fprintf(fp,"----------------------------------------");\
    fprintf(fp,"----------------------------------------\n")
#define RAD_TO_DEG(x)    (180. * x) / PI
#define RECIPROCITY_FLAG    dflags[4]

double detect_range(prt_out, delta_r, limit, pressure, src_level, back_level)
struct dataout prt_out;
complex *pressure;
double delta_r, src_level, back_level;
int limit;
{
    int i, j, flag[5];
    double r, gain, slope, r_last, gain_last, range[5];
    static double prt_gain[5] = {-10., -5., 0., 5., 10.};
    FILE *fp;

/*
 *  Initialize Flag Array
 */
    for (i = 0; i < 5; i++)
    {
        flag[i] = 1;
        range[i] = -999.;
    }
    r = 2. * delta_r;
    gain = 20. * log10(cmod(pressure[2]));
    gain_last = src_level + gain - back_level;
    r_last = r;
    r += delta_r;
    for (i = 3; i <= limit; i++)
    {
/*
```

```
 *      Find Gain Relative To 0 dB at 1 meter (Transmission Loss)
 */
        gain = 20 * log10(cmod(pressure[i]));
/*
 *      Write Out The Levels Versus Range
 */
        gain = src_level + gain - back_level;
        for (j = 0; j < 5; j++)
        {
            if ((gain == prt_gain[j])&&flag[j])
            {
                flag[j] = 0;
                range[j] = r;
            }
            if ((gain < prt_gain[j])&&flag[j])
            {
                slope = (gain_last - gain) / (r_last - r);
                range[j] = r + (prt_gain[j] - gain) / slope;
                flag[j] = 0;
            }
        }
        r_last = r;
        gain_last = gain;
        r += delta_r;
    }
    for (j = 0; j < prt_out.n_fprint; j++)
    {
        for (i = 0; i < 5; i++)
            fprintf(prt_out.fp[j],"% 7.1lf      ",range[i]);
        fprintf(prt_out.fp[j], "\n");
    }
    return gain;
}
```

FFP.C

```
/*
 *   FAST FIELD PROGRAM
 */

#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>
#include "complex.h"
#include "ffp.h"

#define N         2048
#define SQ(X)         (X) * (X)
#define WAVESPECT    dflags[7]

void cutoff();
void fourier();
void profile();
void setup();
void voltage();
void wavenumber();
void zeffective();
double detect_range();
extern int errno;

double propmod(a, prt_out, frequency, src_level, back_level, geometry, ground,
     n_ground_layers, n_interfaces, dflags)
double frequency, src_level, back_level;
int n_ground_layers, n_interfaces, *dflags;
struct geometry *geometry;
struct ground *ground;
struct interface *a;
struct dataout prt_out;
{
     complex amplitude, field[N], impedance, pressure[N], z, z0, z1;
     double delta_k, delta_r, detector, gain, k, k_max, k_min, k_width,
          omega, r, range, source, temp, extra;
     float buffer[2];
     int i, j, limit, n_layers, n_panels, points;
     struct gp gp;
     struct layer *layer;
     FILE *fp_wspc;
```

```
/*
 *   Preliminary Setup
 */
     detector = geometry->Zr;
     omega = TWO_PI * frequency;
     range = geometry->range;
     source = geometry->Zs;
/*
 *   Setup The Points and Panels For Transform
 */
     setup(&n_panels,&points,&extra,dflags);
/*
 *   Impedance and Wavenumber of Half-Spaces #1 and #N
 */
     zeffective(a[1].c, frequency, ground, &impedance, &gp.k_1,
           n_ground_layers, dflags);
     Re(gp.impedance_1) = Re(impedance);
     Im(gp.impedance_1) = -Im(impedance);
     Im(gp.k_1) = -Im(gp.k_1);
     Re(gp.impedance_n) = Re(gp.k_n) = 1;
     Im(gp.impedance_n) = Im(gp.k_n) = 0;
/*
 *   Initialize The Profile
 */
     layer = (struct layer *) calloc((size_t) n_interfaces + 3,
           sizeof(struct layer));
     profile(a, detector, frequency, &gp, layer, n_interfaces, &n_layers,
           source);
/*
 *   Squares of Intrinsic Wave Numbers Within Each Layer
 */
     wavenumber(extra, &gp, layer, n_layers, omega);
/*
 *   Upper Cut-Off Wave Number
 */
     cutoff(&delta_k, &delta_r, frequency, &gp, &k_max, &k_width, layer,
           &n_panels, omega, points, range, dflags);
/*
 *   Zero The Pressure Array
 */
     for (i = 1; i <= points; i++)
           Re(pressure[i]) = Im(pressure[i]) = 0;
/*
 *   z0 = (1 - i) / (2 * sqrt(PI))
```

```
*/
    Re(z0) = ONE_2SQRTPI; Im(z0) = -ONE_2SQRTPI;
/*
 *  Overlap And Add Field From Each Panel
 */
    for (i = 1; i <= n_panels; i++)
    {
/*
 *      Starting Wave Number of Each Panel
 */
        k = k_min = (i - 1) * k_width;
/*
 *      Find The Pressure Field in The Transform Domain
 */
        for (j = 1; j <= points; j++)
        {
/*
 *          Pressure Amplitude
 */
            voltage(&amplitude, &gp, k, layer, n_layers, omega);
            cmul(field[j], amplitude, z0);
            temp = sqrt(k + 1e-20);
            Re(field[j]) /= temp;
            Im(field[j]) /= -temp;
/*
 *              Write Out Horizontal Wave Number Spectrum
 */
                if (WAVESPECT)
                {
                        if ((fp_wspc = fopen("wavnum.e","a")) == NULL)
                        {
                            fprintf(stderr,"error number %d\n",
                        errno);
                            perror("wavnum.e");
                            exit(1);
                        }
                        fprintf(fp_wspc,"%10.4lf\t%10.4lf\t",
                            k,cmod(amplitude));
                        fprintf(fp_wspc,"%10.4lf\t%10.4lf\n",
                            Re(amplitude),Im(amplitude));
                        close(fp_wspc);
                }
            k += delta_k;
        }
/*
 *      Fourier Transform The Wave Number Amplitudes
 */
        fourier(field, points, 1);
/*
 *      Retrieve The Incremental Pressure Amplitudes
 */
        r = delta_r;
        for (j = 2; j <= points; j++)
        {
            Im(field[j]) = -Im(field[j]);
            Re(z1) = extra * sqrt(SQ(r) + SQ(source - detector));
            Im(z1) = -k_min * r;
            cexp(z, z1);
            cmul(amplitude, field[j], z);
            temp = delta_k / sqrt(r);
            Re(amplitude) *= temp;
            Im(amplitude) *= temp;
/*
 *          Accumulate The Total Pressure in Pressure[j]
 */
            Re(pressure[j]) += Re(amplitude);
            Im(pressure[j]) += Im(amplitude);
            if (i == n_panels && r > range)
                break;
            r += delta_r;
        }
    }
    limit = j - 2;
    limit++;
/*
 *  Determine Maximum Detection Range
 */
    gain = detect_range(prt_out, delta_r, limit, pressure, src_level,
        back_level);
    return gain;
}
```

FOURIER.C

```
/*
 *  Replaces data by its discrete Fourier transform if sign == 1 or
 *  by n times its inverse discrete Fourier transform if sign == -1.
 *  data is a complex array of length n.
 *  n must be an integral power of 2 (this is not checked for!).
 *
 *  W. H. Press et al,
 *  Numerical Recipes, Cambridge University, 1986, page 394.
 */

#include "complex.h"

void fourier(data, n, sign)
complex *data;
int n, sign;
{
    complex w, wp, ztemp;
    double temp, theta;
    int index, j = 1, m, max, step;

/*
 *  bit-reversal section
 */
    for (index = 1; index <= n; index++)
    {
        if (j > index)
        {
            Re(ztemp) = Re(data[j]);
            Im(ztemp) = Im(data[j]);
            Re(data[j]) = Re(data[index]);
            Im(data[j]) = Im(data[index]);
            Re(data[index]) = Re(ztemp);
            Im(data[index]) = Im(ztemp);
        }
        m = n / 2;
        while (m >= 2 && m < j)
        {
            j -= m;
            m /= 2;
        }
        j += m;
    }
/*
 *  beginning of Danielson-Lanczos section
 */
    max = 1;
    while (max < n)
    {   /* while loop executed log2(n) times */
/*
 *      initialize for trigonometric recurrence
 */
        step = 2 * max;
        theta = PI / (max * sign);
        temp = sin(theta / 2);
        Re(wp) = -2 * temp * temp;
        Im(wp) = sin(theta);
        Re(w) = 1;
        Im(w) = 0;
/*
 *      two nested inner loops
 */
        for (m = 1; m <= max; m++)
        {
            for (index = m; index <= n; index += step)
            {
/*
 *              Danielson-Lanczos formula
 */
                j = index + max;
                cmul(ztemp, data[j], w);
                csub(data[j], data[index], ztemp);
                cadd(data[index], data[index], ztemp);
            }
/*
 *          trigonometric recurrence
 */
            cmul(ztemp, w, wp);
            cadd(w, w, ztemp);
        }
        max = step;
    }
}
```

HEADER.C

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include "ffp.h"
#include "complex.h"
/*
 *   print a line of 80 dashes
 */
#define dash\
     fprintf(fp,"----------------------------------------");\
     fprintf(fp,"----------------------------------------\n")
#define RAD_TO_DEG(x)    (180. * x) / PI
#define RECIPROCITY_FLAG    dflags[4]
void file_header(prt_out, frequency, geometry, src_level, back_level, dflags)
struct dataout prt_out;
double frequency, src_level, back_level;
int *dflags;
struct geometry *geometry;
{
     FILE *fp;
     int index;

     for (index = 0; index < prt_out.n_fprint; index++)
     {
          fp = prt_out.fp[index];
          fprintf(fp,"\n\n\n                    Scanning Fast Field Program (SCAFFIP)\n");
          fprintf(fp,"                              EOSAEL Version\n");
          dash;
          fprintf(fp,"SOURCE HEIGHT   = %7.1lf METERS          FREQUENCY
               = %7.1lf Hz\n", geometry->Zs, frequency);
          if (RECIPROCITY_FLAG)
               fprintf(fp,"DETECTOR HEIGHT = %7.1lf METERS          SOURCE
                    CENTERED\n", geometry->Zr);
          else
               fprintf(fp,"DETECTOR HEIGHT = %7.1lf METERS
                    DETECTOR CENTERED\n", geometry->Zr);
          fprintf(fp,"SOURCE LEVEL    = %7.1lf DECIBELS          BACKGROUND
               LEVEL = %7.1lf DECIBELS\n",src_level,back_level);
          fprintf(fp,"MAXIMUM RANGE   = %7.1lf METERS\n",geometry->range);
          dash;
          fprintf(fp,"                    DETECTION RANGE (METERS) OVER BENIGN
               TERRAIN\n");
```

```
                dash;
                fprintf(fp,"            ¦                    SIGNAL-NOISE (dB)\n");
                fprintf(fp,"AZIMUTH (DEG) ¦");
                fprintf(fp,"   -10        -5         0         5         10\n");
                dash;
            }
        }
```

INPUT.C

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include "ffp.h"
#include "complex.h"

#define MET_FORM         dflags[1]
#define DEG_TO_RAD(x)    (PI * x) / 180.

extern int errno;

void input(a,dflags,frequency,src_level,back_level,geometry,ground,
    n_ground_layers,n_interfaces,n_freq,prt_out)
int *n_ground_layers, *n_interfaces, *n_freq, *dflags;
double *frequency, *src_level, *back_level;
struct geometry *geometry;
struct ground *ground;
struct interface **a;
struct dataout *prt_out;
{
    FILE *fp;
    int code,index;
    double wind_theta,azmuth1,azmuth2,dazmuth;
    char vdesc[30],vmet[80];
/*
 *  Open Input File
 */
    if ((fp = fopen("scaffip.i", "r")) == NULL)
    {
        fprintf(stderr, "error number %d\n", errno);
        perror("scaffip.i");
        exit(1);
    }
/*
 *  Read Debug Option
 */
    fscanf(fp, "%d", &code);
    fgets(vdesc,30,fp);
    parser(code,dflags);
/*
 *  Read Print Location Option
 */
```

```
        fscanf(fp, "%d", &(prt_out->option));
        fgets(vdesc,30,fp);
/*
 *     Read Geometry Data
 */
        fscanf(fp, "%lf", &(geometry->Zs));
        fgets(vdesc,30,fp);
        fscanf(fp, "%lf", &(geometry->Zr));
        fgets(vdesc,30,fp);
        fscanf(fp, "%lf", &(geometry->range));
        fgets(vdesc,30,fp);
        fscanf(fp, "%lf", &azmuth1);
        fgets(vdesc,30,fp);
        fscanf(fp, "%lf", &azmuth2);
        fgets(vdesc,30,fp);
        fscanf(fp, "%lf", &dazmuth);
        fgets(vdesc,30,fp);
        geometry->theta_1 = DEG_TO_RAD(azmuth1);
        geometry->theta_2 = DEG_TO_RAD(azmuth2);
        geometry->delta_theta = DEG_TO_RAD(dazmuth);
/*
 *     Read Frequency Data
 */
        fscanf(fp, "%lf", frequency);
        fgets(vdesc,30,fp);
        fscanf(fp, "%lf", src_level);
        fgets(vdesc,30,fp);
        fscanf(fp, "%lf", back_level);
        fgets(vdesc,30,fp);
/*
 *     Read Ground Data
 */
        fscanf(fp, "%d", n_ground_layers);
        fgets(vdesc,30,fp);
        fscanf(fp, "%lf", &(ground->sigma_1));
        fgets(vdesc,30,fp);
        fscanf(fp, "%lf", &(ground->omega_1));
        fgets(vdesc,30,fp);
        fscanf(fp, "%lf", &(ground->pn_1));
        fgets(vdesc,30,fp);
        fscanf(fp, "%lf", &(ground->sf_1));
        fgets(vdesc,30,fp);
        if (*n_ground_layers > 1)
        {
/*
```

```
 *      Read Second Ground Data
 */
        fscanf(fp, "%lf", &(ground->depth));
        fgets(vdesc,30,fp);
        fscanf(fp, "%lf", &(ground->sigma_2));
        fgets(vdesc,30,fp);
        fscanf(fp, "%lf", &(ground->omega_2));
        fgets(vdesc,30,fp);
        fscanf(fp, "%lf", &(ground->pn_2));
        fgets(vdesc,30,fp);
        fscanf(fp, "%lf", &(ground->sf_2));
        fgets(vdesc,30,fp);
    }
    fclose(fp);
/*
 *  Read Weather File
 */
    if ((fp = fopen("met.i", "r")) == NULL)
    {
        fprintf(stderr, "error number %d\n", errno);
        perror("met.i");
        exit(1);
    }
    fscanf(fp, "%d", n_interfaces);
    fgets(vdesc,30,fp);
    fgets(vmet,80,fp);
    *a = (struct interface *) calloc((size_t) *n_interfaces + 1,
        sizeof(struct interface));
    if (MET_FORM)
    {
/*
 *      Wind Direction in Radians
 */
        for (index = 1; index <= *n_interfaces; index++)
        {
            fscanf(fp, "%lf %lf %lf %lf %lf %lf",
                &(*a)[index].z, &(*a)[index].kelvin,
                &(*a)[index].relative_humidity,
                &(*a)[index].pressure,
                &(*a)[index].wind_speed,
                &(*a)[index].wind_theta);
        }
    }
    else
    {
```

```
/*
 *      Wind Direction in Degrees
 */
        for (index = 1; index <= *n_interfaces; index++)
        {
            fscanf(fp, "%lf %lf %lf %lf %lf",
                &(*a)[index].z, &(*a)[index].kelvin,
                &(*a)[index].relative_humidity,
                &(*a)[index].pressure,
                &(*a)[index].wind_speed, &wind_theta);
            (*a)[index].wind_theta = DEG_TO_RAD(wind_theta);
        }
    }
    fclose(fp);
}
```

INSERT.C

```
/*
 *   insert a new layer with top interface height == height
 */

#include <stdio.h>
#include <stdio.h>
#include "ffp.h"

void insert(height, index, layer, n_layers, z)
double height, *z;
int *index, *n_layers;
struct layer *layer;
{
/*
 *   find the index of height in z if it exists
 */
    for (*index = 1; *index < *n_layers; ++*index)
        if (z[*index] == height)
        {
/*
 *              return the index of z that yields height
 */
            return;
        }
/*
 *   shift array z up to open a slot for height
 */
    for (*index = *n_layers; *index > 1; --*index)
    {
        layer[*index+1].c = layer[*index].c;
        layer[*index+1].mu = layer[*index].mu;
        layer[*index+1].rho = layer[*index].rho;
        z[*index+1] = z[*index];
/*
 *          check if height belongs above the next z
 */
        if (height > z[*index-1])
            break;
    }
/*
 *   extend the array length and insert height
 */
    ++*n_layers;
    z[*index] = height;
}
```

MAIN.C

```
/*
 *  MASTER.C - Main Driver For General Purpose User Interface
 *
 *  John M. Noble
 *  U.S. Army Research Laboratory
 *  Battlefield Environment Directorate
 *  ATTN:  AMSRL-BE-S
 *  White Sands Missile Range, New Mexico
 *
 *  History:
 *
 *  Dec 1991 - Converted to C by Frank Owens - Ball State Univ.
 *  Feb 1993 - User Interface Generalized
 *  Jul 1993 - Programmed EOSAEL Version of User Interface
 */
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include "ffp.h"
#include "complex.h"

#define FLAGS          O_WRONLY ¦ O_CREAT ¦ O_TRUNC
#define RAD_TO_DEG(x)    (180. * x) / PI

extern int errno;

double propmod();
void file_header();
void input();
void parser();
void sound();

main(argc, argv)
int  argc;
char **argv;
{
    char *id = "@(#)December 1991 version - Frank W. Owens";
    double attenuation, frequency, src_level, back_level, theta;
    float buffer;
    int index,n_ground_layers,n_interfaces,dflags[15],n_freq,j;
    FILE *fp;
    struct geometry geometry;
```

```
        struct ground ground;
        struct interface *a;
        struct dataout prt_out;

        input(&a,dflags,&frequency,&src_level,&back_level,&geometry,
            &ground,&n_ground_layers,&n_interfaces,&n_freq,&prt_out);
/*
 *  Open Output File(s)
 */
        if (prt_out.option != 2)
        {
            if ((prt_out.fp[0] = fopen("scaffip.out", "w")) == NULL)
            {
                fprintf(stderr, "error number %d\n", errno);
                perror("scaffip.out");
                exit(1);
            }
        }
        if (prt_out.option == 2)
            prt_out.fp[0] = stdout;
        else
            prt_out.fp[1] = stdout;
        if (prt_out.option < 3)
            prt_out.n_fprint = 1;
        else
            prt_out.n_fprint = 2;
/*
 *  Print The Header Information
 */
        file_header(prt_out,frequency,&geometry,src_level,back_level,dflags);
/*
 *  Begin Scanning Through The Azimuths
 */
        for (theta = geometry.theta_1; theta < geometry.theta_2 - 0.01;
        theta += geometry.delta_theta)
        {
            for (j = 0; j < prt_out.n_fprint; j++)
                fprintf(prt_out.fp[j], "% 8.1lf\t", RAD_TO_DEG(theta));
            sound(a, n_interfaces, theta, dflags);
            attenuation = propmod(a,prt_out,frequency,src_level,
                back_level,&geometry,&ground,n_ground_layers,
                n_interfaces,dflags);
        }
        for (j = 0; j < prt_out.n_fprint; j++)
        {
            fprintf(prt_out.fp[j],"----------------------------------------");
            fprintf(prt_out.fp[j],"----------------------------------------\n");
        }
        if (prt_out.option != 2)
            fclose(prt_out.fp[0]);
        return 0;
    }
```

PARAMETE.C

```
/*
 *   NETWORK TRANSMISSION LINE PARAMETERS FOR SECTION l_index
 */

#include <stdio.h>
#include "complex.h"
#include "ffp.h"

void parameters(gp, k, l_index, layer, n_layers, omega)
double k, omega;
int l_index, n_layers;
struct gp *gp;
struct layer *layer;
{
    complex z;
/*
 *   electrical wave number kz == - j * gamma
 */
    Re(z) = Re(layer[l_index].ki2) - k * k;
    Im(z) = Im(layer[l_index].ki2);
    csqrt(layer[l_index].kz, z);
/*
 *   Re(gamma) is positive for the correct branch
 */
    if (Im(layer[l_index].kz) > 0)
    {
        Re(layer[l_index].kz) = -Re(layer[l_index].kz);
        Im(layer[l_index].kz) = -Im(layer[l_index].kz);
    }
/*
 *   electrical characteristic admittance
 */
    Re(layer[l_index].y0) =
        Re(layer[l_index].kz) / (omega * layer[l_index].rho);
    Im(layer[l_index].y0) =
        Im(layer[l_index].kz) / (omega * layer[l_index].rho);
/*
 *   admittance for half-spaces
 */
    if (l_index == 1)
    {
        cmul(z, gp->impedance_1, gp->k_1);
```

```
            cdiv(layer[l_index].y0, layer[l_index].y0, z);
        }
        if (l_index == n_layers)
        {
            cmul(z, gp->impedance_n, gp->k_n);
            cdiv(layer[l_index].y0, layer[l_index].y0, z);
        }
    }
```

PARSER.C

```
#include <stdio.h>

void parser(code,dflags)
int code, *dflags;
{
/*
 *   DFLAGS(0) - 2^0 = 1
 */
     dflags[0] = (1 & code);
/*
 *   DFLAGS(1) - 2^1 = 2
 */
     dflags[1] = (2 & code) / 2;
/*
 *   DFLAGS(2) - 2^2 = 4
 */
     dflags[2] = (4 & code) / 4;
/*
 *   DFLAGS(3) - 2^3 = 8
 */
     dflags[3] = (8 & code) / 8;
/*
 *   DFLAGS(4) - 2^4 = 16
 */
     dflags[4] = (16 & code) / 16;
/*
 *   DFLAGS(5) - 2^5 = 32
 */
     dflags[5] = (32 & code) / 32;
/*
 *   DFLAGS(6) - 2^6 = 64
 */
     dflags[6] = (64 & code) / 64;
/*
 *   DFLAGS(7) - 2^7 = 128
 */
     dflags[7] = (128 & code) / 128;
/*
 *   DFLAGS(8) - 2^8 = 256
 */
     dflags[8] = (256 & code) / 256;
/*
```

```
 *  DFLAGS(9) - 2^9 = 512
 */
    dflags[9] = (512 & code) / 512;
/*
 *  DFLAGS(10) - 2^10 = 1024
 */
    dflags[10] = (1024 & code) / 1024;
/*
 *  DFLAGS(11) - 2^11 = 2048
 */
    dflags[11] = (2048 & code) / 2048;
/*
 *  DFLAGS(12) - 2^12 = 4096
 */
    dflags[12] = (4096 & code) / 4096;
/*
 *  DFLAGS(13) - 2^13 = 8192
 */
    dflags[13] = (8192 & code) / 8192;
/*
 *  DFLAGS(14) - 2^14 = 16384
 */
    dflags[14] = (16384 & code) / 16384;
}
```

PROFILE.C

```
/*
 *  INITIALIZE PROBLEM ATMOSPHERIC PROFILE
 */

#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>
#include "ffp.h"

double air_absorption();
void insert();

void profile(a, detector, frequency, gp, layer, n_interfaces, n_layers, source)
double detector, frequency, source;
int n_interfaces, *n_layers;
struct interface *a;
struct gp *gp;
struct layer *layer;
{
    double *z;
    int index;

/*
 *  initialize the problem profile (z increases)
 */
    z = (double *) calloc((size_t) n_interfaces + 3, sizeof(double));
    for (index = 1; index <= n_interfaces; index++)
    {
        layer[index].c = a[index].c;
        layer[index].mu = air_absorption(frequency, a[index].kelvin,
            a[index].pressure, a[index].relative_humidity);
        layer[index].rho = 1.2;
        z[index] = a[index].z;
    }
    *n_layers = n_interfaces;
/*
 *  insert detector and source interfaces (lowest first)
 */
    if (source < detector)
    {
        insert(source, &gp->index_source, layer, n_layers, z);
        insert(detector, &gp->index_detector, layer, n_layers, z);
        }
        else
        {
            insert(detector, &gp->index_detector, layer, n_layers, z);
            insert(source, &gp->index_source, layer, n_layers, z);
        }
    /*
     *  convert interface heights to layer thicknesses
     */
        for (index = 2; index < *n_layers; index++)
            layer[index].thickness = z[index] - z[index-1];
    }
```

SETUP.C

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>

#define DEFAULT_EXTRA_LOSS        1e-4
#define EXTRA_LOSS_FLAG           dflags[3]
#define POINTS                    1024
#define N_PANELS                  -2
#define TRANSFORM_FLAG            dflags[4]

extern int errno;

void setup(n_panels,points,extra,dflags)
double *extra;
int *n_panels, *points, *dflags;
{
    FILE *fp;
    char filename[81];

/*
 *  Extra Loss
 */
    if (EXTRA_LOSS_FLAG)
    {
        if ((fp = fopen("extra.los", "r")) == NULL)
        {
            fprintf(stderr, "error number %d\n", errno);
            perror("extra.los");
            exit(1);
        }
        fscanf(fp, "%lf", extra);
        fclose(fp);
    }
    else
        *extra = DEFAULT_EXTRA_LOSS;
/*
 *  Number of Points and Panels
 */
    if (TRANSFORM_FLAG)
    {
        if ((fp = fopen("trans.frm", "r")) == NULL)
        {
                fprintf(stderr, "error number %d\n", errno);
                perror("trans.frm");
                exit(1);
            }
            fscanf(fp, "%lf %lf", points,n_panels);
            fclose(fp);
        }
        else
        {
            *points = POINTS;
            *n_panels = N_PANELS;
        }
    }
```

SOUND.C

```
/*
 *   calculate the sound speed profile
 */

#include <stdio.h>
#include "complex.h"
#include "ffp.h"

#define T0   273.16        /* Kelvin freezing point of H2O */
#define R0   8314.16            /* universal gas constant        */
#define SOUND_SPEED_OUT_FLAG      dflags[0]
#define RECIPROCITY_FLAG          dflags[2]

extern int errno;

void sound(a, n_interfaces, bearing, dflags)
double bearing;
int n_interfaces, *dflags;
struct interface *a;
{
    FILE *fp_sound;
    int index;
    double gamma, humidity, lpsat, R, temp, theta;

    for (index = 1; index <= n_interfaces; index++)
    {
/*
 *      Angle Between Wind And Propagation Path
 */
        if (RECIPROCITY_FLAG)
        {
/*
 *          Source Centered Geometry
 */
            theta = a[index].wind_theta - PI - bearing;
            }
        else
        {
/*
 *          Sensor Centered Geometry
 */
            theta = a[index].wind_theta - bearing;
            }
/*
 *          The Partial Pressure of Saturated Water Vapor
 */
            temp = a[index].kelvin / T0;
            lpsat = -10.79586 / temp - 5.02808 * log10(temp)
                - 1.50474e-4 * pow(10.0, 8.29692 * (1 - temp))
                + 4.28730e-4 * pow(10.0, 4.76955 * (1 - temp))
                + 8.575983444;
/*
 *          The Fraction of Water Molecules in Air
 */
            humidity = a[index].relative_humidity
                * pow(10.0, (lpsat - 2)) / a[index].pressure;
/*
 *          Ratio of Specific Heats
 */
            gamma = (humidity + 7) / (humidity + 5);
            R = R0 / (29 - 11 * humidity);
/*
 *          Sound Speed Due to Humidity And Temperature
 */
            a[index].c = sqrt(a[index].kelvin * gamma * R);
/*
 *          Adjustment For Wind
 */
            a[index].c += a[index].wind_speed * cos(theta);
        }
/*
 *      Print Sound Speed Profile If Desired
 */
        if (SOUND_SPEED_OUT_FLAG)
        {
            if ((fp_sound = fopen("sound.o","a")) == NULL)
            {
                fprintf(stderr,"error number %d\n",errno);
                perror("sound.o");
                exit(1);
            }
            for (index = 1; index <= n_interfaces; index++)
                fprintf(fp_sound,"%10.4lf\t%10.4lf\n",
                    a[index].z,a[index].c);
            fclose(fp_sound);
        }
}
```

VOLTAGE.C

```
/*
 *  FINDS THE VOLTAGE detector_voltage AT index_detector
 *  DUE TO CURRENT SOURCE source_current AT index_source
 */

#include <stdio.h>
#include "complex.h"
#include "ffp.h"

void admit();
void clip();

void voltage(detector_voltage, gp, k, layer, n_layers, omega)
complex *detector_voltage;
double k, omega;
int n_layers;
struct gp *gp;
struct layer *layer;
{
    complex admittance_1, admittance_n, source_admittance,
        source_current, source_voltage, v;
    int last_1, last_n;

/*
 *  zero default voltage
 */
    Re(*detector_voltage) = Im(*detector_voltage) = Re(v) = Im(v) = 0;
/*
 *  find last significant line toward half-space #N
 */
    clip(gp, k, &last_n, layer, n_layers, n_layers, omega);
    if (last_n != -1)
    {       /* if (v == 0) return zero volts */
/*
 *      find last significant line toward half-space #1
 */
        clip(gp, k, &last_1, layer, 1, n_layers, omega);
        if (last_1 != -1)
        {   /* if (v == 0) return zero volts */
/*
 *          admittance that the source sees looking toward #N
 */
```

```
                admit(&admittance_n, gp, last_n, layer, &v);
/*
 *              admittance that the source sees looking toward #1
 */
                admit(&admittance_1, gp, last_1, layer, &v);
/*
 *              total source admittance (add in parallel)
 */
                cadd(source_admittance, admittance_1, admittance_n);
/*
 *              source current (equation 14)
 */
                Re(source_current) = 2 * k /
                    (layer[gp->index_source].rho * omega);
                Im(source_current) = 0;
/*
 *              source voltage (equation 20)
 */
                cdiv(source_voltage, source_current, source_admittance);
/*
 *              detector voltage (equation 23)
 */
                cdiv(*detector_voltage, source_voltage, v);
            }
        }
}
```

WAVENUMB.C

```
/*
 *  SQUARES OF COMPLEX INTRINSIC WAVENUMBERS FOR EACH LINE
 *  SECTION
 */

#include <stdio.h>
#include "complex.h"
#include "ffp.h"

void wavenumber(extra, gp, layer, n_layers, omega)
double extra, omega;
int n_layers;
struct gp *gp;
struct layer *layer;
{
    complex z;
    int index;
/*
 *  finite thickness slabs
 */
    for (index = 2; index < n_layers; index++)
    {
        Re(z) = omega / layer[index].c;
        Im(z) = - extra - layer[index].mu;
        cmul(layer[index].ki2, z, z);
    }
/*
 *  half-spaces #1 and #N
 */
    Re(z) = omega / layer[1].c * Re(gp->k_1);
    Im(z) = omega / layer[1].c * Im(gp->k_1) - extra - layer[1].mu;
    cmul(layer[1].ki2, z, z);
    Re(z) = omega / layer[n_layers].c * Re(gp->k_n);
    Im(z) = omega / layer[n_layers].c * Im(gp->k_n)
        - extra - layer[n_layers].mu;
    cmul(layer[n_layers].ki2, z, z);
}
```

Z2LAYERS.C

```
/*
 *  FIND THE IMPEDANCE FOR ONE OR TWO GROUND LAYERS
 *
 *  K. Attenborough,
 *  Acoustical Impedance Models for Outdoor Ground Surfaces,
 *  J. Sound Vib. 99 (1985), 521-544.
 *
 *  See equation 15 on page 527 of the above reference.
 */
#include "complex.h"

void z2layers(depth, impedance, k1, zbottom, ztop)
complex *impedance, *k1, zbottom, ztop;
double depth;
{
     complex factor, z, z1, z2, ztan;
     double a, b;

/*
 *  Zb = zbottom
 *  Zt = ztop
 *              Zb - i Zt tan(a + ib)      z1
 *  impedance = -------------------- Zt = -- Zt = factor Zt
 *              Zt - i Zb tan(a + ib)      z2
 */
     a = depth * Re(*k1);
     b = depth * Im(*k1);
/*
 *  tan(a + ib) = (tan(a) + i tanh(b)) / (1 - i tan(a)tanh(b))
 */
     Re(z1) = tan(a);
     Im(z1) = tanh(b);
     Re(z2) = 1;
     Im(z2) = -Re(z1) * Im(z1);
     cdiv(ztan, z1, z2);
/*
 *  determine the factor
 */
     cmul(z, ztop, ztan);
     Re(z1) = Re(zbottom) + Im(z);
     Im(z1) = Im(zbottom) - Re(z);
     cmul(z, zbottom, ztan);
```

```
        Re(z2) = Re(ztop) + Im(z);
        Im(z2) = Im(ztop) - Re(z);
        cdiv(factor, z1, z2);
    /*
     *  complex impedance
     */
        cmul(*impedance, factor, ztop);
    }
```

Z4PARAME.C

```
/*
 *  FOUR-PARAMETER APPROXIMATE MODEL FOR GROUND IMPEDANCE
 *
 *  K. Attenborough,
 *  Acoustical Impedance Models for Outdoor Ground Surfaces,
 *  J. Sound Vib. 99 (1985), 521-544.
 */

#include "complex.h"

#define GAMMA      1.4
#define RHO0    1.21       /* k / m^3 */

void z4parameter(c, frequency, impedance, wavenumber, omega, pn, sf, sigma)
complex *impedance, *wavenumber;
double c, frequency, omega, pn, sf, sigma;
{
     complex kb, ztemp;
     double alpha, beta, q2, x;

     alpha = TWO_PI * frequency;
     beta = alpha / c;
/*
 *  1000 converts from cgs to mks
 */
     x = 1000 * sf * sf * sigma / (RHO0 * alpha);
/*
 *  q2 defined after equation 8 on page 524
 */
     q2 = pow(omega, -pn);
/*
 *  kb defined by equation 11 on page 524
 */
     Re(ztemp) = 1.5764 * q2;
     Im(ztemp) = GAMMA * omega * x;
     csqrt(kb, ztemp);
/*
 *  impedance defined by equation 12 on page 524
 */
     Re(ztemp) = (4 * q2) / (3 * omega);
     Im(ztemp) = x;
     cdiv(*impedance, ztemp, kb);
/*
 *   kb has no units
 */
    Re(*wavenumber) = beta * Re(kb);
    Im(*wavenumber) = beta * Im(kb);
}
```

ZEFFECTI.C

```
/*
 *  CALCULATES EFFECTIVE GROUND IMPEDANCE
 */
#include <stdio.h>
#include <stddef.h>
#include "complex.h"
#include "ffp.h"

#define PRT_IMPED      dflags[6]

void z2layers();
void z4parameter();
extern int errno;

void zeffective(c, frequency, ground, impedance, k1, n_ground_layers, dflags)
complex *impedance, *k1;
double c, frequency;
int n_ground_layers, *dflags;
struct ground *ground;
{
    complex k2, zbottom, ztop;
    FILE *fp_imp;

/*
 *  find impedance for top ground interface
 */
    z4parameter(c, frequency, &ztop, k1,
        ground->omega_1, ground->pn_1,
        ground->sf_1, ground->sigma_1);
    Re(*impedance) = Re(ztop);
    Im(*impedance) = Im(ztop);
    if (n_ground_layers == 2)
    {
/*
 *      put in the contribution from the bottom ground interface
 */
        z4parameter(c, frequency, &zbottom, &k2,
            ground->omega_2, ground->pn_2,
            ground->sf_2, ground->sigma_2);
        z2layers(ground->depth, impedance, k1, zbottom, ztop);
    }
/*
```

```
 *      Write Out Impedance Value
 */
        if (PRT_IMPED)
        {
                if ((fp_imp = fopen("imped.e","a")) == NULL)
                {
                        fprintf(stderr,"error number %d\n",errno);
                        perror("imped.e");
                        exit(1);
                }
                fprintf(fp_imp,"Impedance = (%10.4lf,%10.4lf)\n",
                        Re(*impedance),-Im(*impedance));
                fprintf(fp_imp,"Wavenumber = (%10.4lf,%10.4lf)\n",
                        Re(*k1),-Im(*k1));
                close(fp_imp);

        }
}
```

COMPLEX.H

```
#include <math.h>

#define ONE_2SQRTPI   0.28209479177387814347
#define PI            3.14159265358979323846
#define TWO_PI        6.28318530717958647692

#ifndef _COMPLEX_
#define _COMPLEX_

typedef struct
{
    double real, imaginary;
} complex;

/*
 *  Re(z) is the real part of z
 *  Im(z) is the imaginary part of z
 */
#define Re(z) (z).real
#define Im(z) (z).imaginary

/*
 *  z is the complex sum of z1 and z2
 */
#define cadd(z, z1, z2)\
    Re(z) = Re(z1) + Re(z2);\
    Im(z) = Im(z1) + Im(z2)

/*
 *  z is the complex quotient of z1 and z2
 *       z = z1 / z2
 */
#define cdiv(z, z1, z2)\
{\
    double denominator, temp1, temp2;\
\
    denominator = Re(z2) * Re(z2) + Im(z2) * Im(z2);\
    temp1 = Im(z1) * Re(z2);\
    temp2 = Re(z1) * Im(z2);\
    Re(z) = ((Re(z1) + Im(z1)) * (Re(z2) + Im(z2))\
        - temp1 - temp2) / denominator;\
    Im(z) = (temp1 - temp2) / denominator;\
}

/*
 *   z is the complex exponential of z1
 */
#define cexp(z, z1)\
{\
    double r, theta;\
\
    r = exp(Re(z1));\
    theta = fmod(Im(z1), TWO_PI);\
    Re(z) = r * cos(theta);\
    Im(z) = r * sin(theta);\
}

/*
 *   cmod(z) returns |z|
 */
#define cmod(z) sqrt(Re(z) * Re(z) + Im(z) * Im(z))

/*
 *   z is the complex product of z1 and z2
 */
#define cmul(z, z1, z2)\
{\
    double temp1, temp2;\
\
    temp1 = Re(z1) * Im(z2);\
    temp2 = Im(z1) * Re(z2);\
    Re(z) = (Re(z1) + Im(z1)) * (Re(z2) - Im(z2)) + temp1 - temp2;\
    Im(z) = temp1 + temp2;\
}

/*
 *   z is the principal square root of z1
 */
#define csqrt(z, z1)\
{\
    double a, b, r;\
\
    a = Re(z1);\
    b = Im(z1);\
    r = sqrt(a * a + b * b);\
/*\
 *  check for truncation error\
 */\
    if ((Re(z) = (r + a) / 2) > 0)\
        Re(z) = sqrt(Re(z));\
    else\
        Re(z) = 0;\
    if ((Im(z) = (r - a) / 2) > 0)\
        Im(z) = sqrt(Im(z));\
    else\
        Im(z) = 0;\
/*\
 *  select the principal branch\
 */\
    if (b < 0)\
        Im(z) = -Im(z);\
}
/*
 *  z is the complex difference of z1 and z2
 */
#define csub(z, z1, z2)\
    Re(z) = Re(z1) - Re(z2);\
    Im(z) = Im(z1) - Im(z2)

#endif
```

FFP.H

```
#ifndef _COMPLEX_
#define _COMPLEX_
typedef struct
{
    double real, imaginary;
} complex;
#endif

struct geometry
{
    double delta_theta, range, theta_1, theta_2, Zr, Zs;
};

struct ground
{
    double depth, omega_1, omega_2, pn_1, pn_2,
        sf_1, sf_2, sigma_1, sigma_2;
};

struct interface
{
    double c, kelvin, pressure, relative_humidity,
        wind_speed, wind_theta, z;
};

struct gp
{
    complex impedance_1, impedance_n, k_1, k_n;
    int index_detector, index_source;
};

struct layer
{
    complex ki2, kz, y0;
    double c, mu, rho, thickness;
};

struct dataout
{
    FILE *fp[2];
    int  n_fprint;
    int  option;
};
```

Appendix B
Example Cases

Example Case #1

Input Files

met.i

```
 61  # of Atmospheric Layers
Height    Temperature     RH        Press      Wind Speed    Wind Dir

  0.000    294.8279      0.00       1.00        0.00          0.00
 25.000    294.8279      0.00       1.00        0.00          0.00
 50.000    294.8279      0.00       1.00        0.00          0.00
 75.000    294.8279      0.00       1.00        0.00          0.00
100.000    294.8279      0.00       1.00        0.00          0.00
125.000    294.8279      0.00       1.00        0.00          0.00
150.000    294.8279      0.00       1.00        0.00          0.00
175.000    294.8279      0.00       1.00        0.00          0.00
200.000    294.8279      0.00       1.00        0.00          0.00
225.000    294.8279      0.00       1.00        0.00          0.00
250.000    294.8279      0.00       1.00        0.00          0.00
275.000    294.8279      0.00       1.00        0.00          0.00
300.000    294.8279      0.00       1.00        0.00          0.00
325.000    294.8279      0.00       1.00        0.00          0.00
350.000    294.8279      0.00       1.00        0.00          0.00
375.000    294.8279      0.00       1.00        0.00          0.00
400.000    294.8279      0.00       1.00        0.00          0.00
425.000    294.8279      0.00       1.00        0.00          0.00
450.000    294.8279      0.00       1.00        0.00          0.00
475.000    294.8279      0.00       1.00        0.00          0.00
500.000    294.8279      0.00       1.00        0.00          0.00
525.000    294.8279      0.00       1.00        0.00          0.00
550.000    294.8279      0.00       1.00        0.00          0.00
575.000    294.8279      0.00       1.00        0.00          0.00
600.000    294.8279      0.00       1.00        0.00          0.00
625.000    294.8279      0.00       1.00        0.00          0.00
650.000    294.8279      0.00       1.00        0.00          0.00
675.000    294.8279      0.00       1.00        0.00          0.00
700.000    294.8279      0.00       1.00        0.00          0.00
725.000    294.8279      0.00       1.00        0.00          0.00
750.000    294.8279      0.00       1.00        0.00          0.00
775.000    294.8279      0.00       1.00        0.00          0.00
800.000    294.8279      0.00       1.00        0.00          0.00
825.000    294.8279      0.00       1.00        0.00          0.00
850.000    294.8279      0.00       1.00        0.00          0.00
875.000    294.8279      0.00       1.00        0.00          0.00
900.000    294.8279      0.00       1.00        0.00          0.00
925.000    294.8279      0.00       1.00        0.00          0.00
950.000    294.8279      0.00       1.00        0.00          0.00
975.000    294.8279      0.00       1.00        0.00          0.00
```

```
1000.000      294.8279    0.00     1.00     0.00     0.00
1025.000      294.8279    0.00     1.00     0.00     0.00
1050.000      294.8279    0.00     1.00     0.00     0.00
1075.000      294.8279    0.00     1.00     0.00     0.00
1100.000      294.8279    0.00     1.00     0.00     0.00
1125.000      294.8279    0.00     1.00     0.00     0.00
1150.000      294.8279    0.00     1.00     0.00     0.00
1175.000      294.8279    0.00     1.00     0.00     0.00
1200.000      294.8279    0.00     1.00     0.00     0.00
1225.000      294.8279    0.00     1.00     0.00     0.00
1250.000      294.8279    0.00     1.00     0.00     0.00
1275.000      294.8279    0.00     1.00     0.00     0.00
1300.000      294.8279    0.00     1.00     0.00     0.00
1325.000      294.8279    0.00     1.00     0.00     0.00
1350.000      294.8279    0.00     1.00     0.00     0.00
1375.000      294.8279    0.00     1.00     0.00     0.00
1400.000      294.8279    0.00     1.00     0.00     0.00
1425.000      294.8279    0.00     1.00     0.00     0.00
1450.000      294.8279    0.00     1.00     0.00     0.00
1475.000      294.8279    0.00     1.00     0.00     0.00
1500.000      294.8279    0.00     1.00     0.00     0.00
```

scaffip.i

```
0             Debug Option
3             Print Option
5.0           Source Height
1.0           Receiver Height
10000.0       Horizontal Range
0.0           Init. Scan Angle
45.0          Final Scan Angle
15.0          Angular Resolution
100.0         Frequency
130.0         Source Level
30.0          Background Level
1             # of Ground Layers
366.          Flow Resistivity
0.27          Omega
0.5           Pore Shape Factor
0.5           Grain Shape Factor
```

Output File

scaffip.o

```
                    Scanning Fast Field Program (SCAFFIP)
                               EOSAEL Version
--------------------------------------------------------------------------------
SOURCE HEIGHT    =  5.0 METERS          FREQUENCY          = 100.0 Hz
DETECTOR HEIGHT  =  1.0 METERS          DETECTOR CENTERED
SOURCE LEVEL     =  130.0 DECIBELS      BACKGROUND LEVEL   = 30.0 DECIBELS
MAXIMUM RANGE    =  10000.0 METERS
--------------------------------------------------------------------------------
             DETECTION RANGE (METERS) OVER BENIGN TERRAIN
--------------------------------------------------------------------------------
               |                  SIGNAL-NOISE (dB)
AZIMUTH (DEG)|     -10         -5          0           5           10
--------------------------------------------------------------------------------
     0.0          5928.1      4893.9      4002.4      3266.5      2657.8
    15.0          5928.1      4893.9      4002.4      3266.5      2657.8
    30.0          5928.1      4893.9      4002.4      3266.5      2657.8
--------------------------------------------------------------------------------
```

Example Case #2

met.i

61 # of Atmospheric Layers

Height	Temperature	RH	Press	Wind Speed	Wind Dir
0.000	294.8279	0.00	1.00	0.00	0.00
25.000	299.1287	0.00	1.00	0.00	0.00
50.000	303.4607	0.00	1.00	0.00	0.00
75.000	307.8239	0.00	1.00	0.00	0.00
100.000	312.2182	0.00	1.00	0.00	0.00
125.000	316.6436	0.00	1.00	0.00	0.00
150.000	321.1002	0.00	1.00	0.00	0.00
175.000	325.5879	0.00	1.00	0.00	0.00
200.000	330.1067	0.00	1.00	0.00	0.00
225.000	334.6567	0.00	1.00	0.00	0.00
250.000	339.2379	0.00	1.00	0.00	0.00
275.000	343.8502	0.00	1.00	0.00	0.00
300.000	348.4936	0.00	1.00	0.00	0.00
325.000	353.1682	0.00	1.00	0.00	0.00
350.000	357.8739	0.00	1.00	0.00	0.00
375.000	362.6108	0.00	1.00	0.00	0.00
400.000	367.3788	0.00	1.00	0.00	0.00
425.000	372.1779	0.00	1.00	0.00	0.00
450.000	377.0082	0.00	1.00	0.00	0.00
475.000	381.8697	0.00	1.00	0.00	0.00
500.000	386.7622	0.00	1.00	0.00	0.00
525.000	391.6860	0.00	1.00	0.00	0.00
550.000	396.6408	0.00	1.00	0.00	0.00
575.000	401.6268	0.00	1.00	0.00	0.00
600.000	406.6440	0.00	1.00	0.00	0.00
625.000	411.6923	0.00	1.00	0.00	0.00
650.000	416.7717	0.00	1.00	0.00	0.00
675.000	421.8823	0.00	1.00	0.00	0.00
700.000	427.0240	0.00	1.00	0.00	0.00
725.000	432.1969	0.00	1.00	0.00	0.00
750.000	437.4009	0.00	1.00	0.00	0.00
775.000	442.6361	0.00	1.00	0.00	0.00
800.000	447.9024	0.00	1.00	0.00	0.00
825.000	453.1998	0.00	1.00	0.00	0.00
850.000	458.5284	0.00	1.00	0.00	0.00
875.000	463.8881	0.00	1.00	0.00	0.00
900.000	469.2790	0.00	1.00	0.00	0.00
925.000	474.7010	0.00	1.00	0.00	0.00
950.000	480.1541	0.00	1.00	0.00	0.00
975.000	485.6384	0.00	1.00	0.00	0.00
1000.000	491.1539	0.00	1.00	0.00	0.00
1025.000	496.7005	0.00	1.00	0.00	0.00
1050.000	502.2782	0.00	1.00	0.00	0.00
1075.000	507.8871	0.00	1.00	0.00	0.00

1100.000	513.5271	0.00	1.00	0.00	0.00
1125.000	519.1982	0.00	1.00	0.00	0.00
1150.000	524.9005	0.00	1.00	0.00	0.00
1175.000	530.6340	0.00	1.00	0.00	0.00
1200.000	536.3985	0.00	1.00	0.00	0.00
1225.000	542.1943	0.00	1.00	0.00	0.00
1250.000	548.0211	0.00	1.00	0.00	0.00
1275.000	553.8792	0.00	1.00	0.00	0.00
1300.000	559.7683	0.00	1.00	0.00	0.00
1325.000	565.6886	0.00	1.00	0.00	0.00
1350.000	571.6401	0.00	1.00	0.00	0.00
1375.000	577.6227	0.00	1.00	0.00	0.00
1400.000	583.6364	0.00	1.00	0.00	0.00
1425.000	589.6813	0.00	1.00	0.00	0.00
1450.000	595.7573	0.00	1.00	0.00	0.00
1475.000	601.8644	0.00	1.00	0.00	0.00
1500.000	608.0027	0.00	1.00	0.00	0.00

scaffip.i

0	Debug Option
3	Print Option
5.0	Source Height
1.0	Receiver Height
10000.0	Horizontal Range
0.0	Init. Scan Angle
45.0	Final Scan Angle
15.0	Angular Resolution
100.0	Frequency
130.0	Source Level
30.0	Background Level
1	# of Ground Layers
366.	Flow Resistivity
0.27	Omega
0.5	Pore Shape Factor
0.5	Grain Shape Factor

Output File

scaffip.o

```
                    Scanning Fast Field Program (SCAFFIP)
                              EOSAEL Version
--------------------------------------------------------------------------------
SOURCE HEIGHT    =  5.0 METERS          FREQUENCY          = 100.0 Hz
DETECTOR HEIGHT  =  1.0 METERS          DETECTOR CENTERED
SOURCE LEVEL     =  130.0 DECIBELS      BACKGROUND LEVEL   = 30.0 DECIBELS
MAXIMUM RANGE    =  10000.0 METERS
--------------------------------------------------------------------------------
              DETECTION RANGE (METERS) OVER BENIGN TERRAIN
--------------------------------------------------------------------------------
                |                 SIGNAL-NOISE (dB)
AZIMUTH (DEG)|     -10         -5          0           5          10
--------------------------------------------------------------------------------
      0.0        9185.6      8746.3      7542.9      6084.5      3412.5
     15.0        9185.6      8746.3      7542.9      6084.5      3412.5
     30.0        9185.6      8746.3      7542.9      6084.5      3412.5
--------------------------------------------------------------------------------
```

Example Case #3

met.i

61 # of Atmospheric Layers

Height	Temperature	RH	Press	Wind Speed	Wind Dir
0.000	294.8279	0.00	1.00	0.00	0.00
25.000	290.5581	0.00	1.00	0.00	0.00
50.000	286.3196	0.00	1.00	0.00	0.00
75.000	282.1121	0.00	1.00	0.00	0.00
100.000	277.9358	0.00	1.00	0.00	0.00
125.000	273.7907	0.00	1.00	0.00	0.00
150.000	269.6767	0.00	1.00	0.00	0.00
175.000	265.5938	0.00	1.00	0.00	0.00
200.000	261.5421	0.00	1.00	0.00	0.00
225.000	257.5216	0.00	1.00	0.00	0.00
250.000	253.5321	0.00	1.00	0.00	0.00
275.000	249.5738	0.00	1.00	0.00	0.00
300.000	245.6467	0.00	1.00	0.00	0.00
325.000	241.7507	0.00	1.00	0.00	0.00
350.000	237.8858	0.00	1.00	0.00	0.00
375.000	234.0521	0.00	1.00	0.00	0.00
400.000	230.2496	0.00	1.00	0.00	0.00
425.000	226.4781	0.00	1.00	0.00	0.00
450.000	222.7378	0.00	1.00	0.00	0.00
475.000	219.0287	0.00	1.00	0.00	0.00
500.000	215.3507	0.00	1.00	0.00	0.00
525.000	211.7038	0.00	1.00	0.00	0.00
550.000	208.0881	0.00	1.00	0.00	0.00
575.000	204.5036	0.00	1.00	0.00	0.00
600.000	200.9501	0.00	1.00	0.00	0.00
625.000	197.4279	0.00	1.00	0.00	0.00
650.000	193.9367	0.00	1.00	0.00	0.00
675.000	190.4767	0.00	1.00	0.00	0.00
700.000	187.0479	0.00	1.00	0.00	0.00
725.000	183.6502	0.00	1.00	0.00	0.00
750.000	180.2836	0.00	1.00	0.00	0.00
775.000	176.9482	0.00	1.00	0.00	0.00
800.000	173.6439	0.00	1.00	0.00	0.00
825.000	170.3707	0.00	1.00	0.00	0.00
850.000	167.1287	0.00	1.00	0.00	0.00
875.000	163.9179	0.00	1.00	0.00	0.00
900.000	160.7382	0.00	1.00	0.00	0.00
925.000	157.5896	0.00	1.00	0.00	0.00
950.000	154.4722	0.00	1.00	0.00	0.00
975.000	151.3859	0.00	1.00	0.00	0.00
1000.000	148.3308	0.00	1.00	0.00	0.00
1025.000	145.3068	0.00	1.00	0.00	0.00
1050.000	142.3139	0.00	1.00	0.00	0.00
1075.000	139.3522	0.00	1.00	0.00	0.00

```
     1100.000         136.4217        0.00         1.00         0.00          0.00
     1125.000         133.5222        0.00         1.00         0.00          0.00
     1150.000         130.6540        0.00         1.00         0.00          0.00
     1175.000         127.8168        0.00         1.00         0.00          0.00
     1200.000         125.0108        0.00         1.00         0.00          0.00
     1225.000         122.2360        0.00         1.00         0.00          0.00
     1250.000         119.4923        0.00         1.00         0.00          0.00
     1275.000         116.7797        0.00         1.00         0.00          0.00
     1300.000         114.0983        0.00         1.00         0.00          0.00
     1325.000         111.4480        0.00         1.00         0.00          0.00
     1350.000         108.8289        0.00         1.00         0.00          0.00
     1375.000         106.2409        0.00         1.00         0.00          0.00
     1400.000         103.6841        0.00         1.00         0.00          0.00
     1425.000         101.1583        0.00         1.00         0.00          0.00
     1450.000          98.6638        0.00         1.00         0.00          0.00
     1475.000          96.2004        0.00         1.00         0.00          0.00
     1500.000          93.7681        0.00         1.00         0.00          0.00
```

scaffip.i

```
0              Debug Option
3              Print Option
5.0            Source Height
1.0            Receiver Height
10000.0        Horizontal Range
0.0            Init. Scan Angle
45.0           Final Scan Angle
15.0           Angular Resolution
100.0          Frequency
130.0          Source Level
30.0           Background Level
1              # of Ground Layers
366.           Flow Resistivity
0.27           Omega
0.5            Pore Shape Factor
0.5            Grain Shape Factor
```

Output File

scaffip.o

```
                    Scanning Fast Field Program (SCAFFIP)
                              EOSAEL Version
--------------------------------------------------------------------------------
SOURCE HEIGHT    =  5.0 METERS          FREQUENCY          = 100.0 Hz
DETECTOR HEIGHT  =  1.0 METERS          DETECTOR CENTERED
SOURCE LEVEL     =  130.0 DECIBELS      BACKGROUND LEVEL  =  30.0 DECIBELS
MAXIMUM RANGE    =  10000.0 METERS
--------------------------------------------------------------------------------
              DETECTION RANGE (METERS) OVER BENIGN TERRAIN
--------------------------------------------------------------------------------
                |                  SIGNAL-NOISE (dB)
AZIMUTH (DEG)   |    -10        -5          0          5          10
--------------------------------------------------------------------------------
     0.0           4919.7     4207.2      2411.9     2400.0      1402.7
    15.0           4919.7     4207.2      2411.9     2400.0      1402.7
    30.0           4919.7     4207.2      2411.9     2400.0      1402.7
--------------------------------------------------------------------------------
```

Example Case #4

met.i

65 # of Atmospheric Layers

Height	Temperature	RH	Press	Wind Speed	Wind Dir
0.000	294.8279	0.00	1.00	0.00	0.00
5.000	295.6855	0.00	1.00	0.00	0.00
10.000	296.5445	0.00	1.00	0.00	0.00
15.000	297.4046	0.00	1.00	0.00	0.00
20.000	298.2661	0.00	1.00	0.00	0.00
25.000	299.1287	0.00	1.00	0.00	0.00
30.000	299.9926	0.00	1.00	0.00	0.00
35.000	300.8578	0.00	1.00	0.00	0.00
40.000	301.7242	0.00	1.00	0.00	0.00
45.000	302.5918	0.00	1.00	0.00	0.00
50.000	303.4607	0.00	1.00	0.00	0.00
55.000	304.3309	0.00	1.00	0.00	0.00
60.000	305.2022	0.00	1.00	0.00	0.00
65.000	306.0749	0.00	1.00	0.00	0.00
70.000	306.9487	0.00	1.00	0.00	0.00
75.000	307.8239	0.00	1.00	0.00	0.00
80.000	308.7002	0.00	1.00	0.00	0.00
85.000	309.5778	0.00	1.00	0.00	0.00
90.000	310.4567	0.00	1.00	0.00	0.00
95.000	311.3368	0.00	1.00	0.00	0.00
100.000	312.2182	0.00	1.00	0.00	0.00
110.000	310.4567	0.00	1.00	0.00	0.00
120.000	308.7002	0.00	1.00	0.00	0.00
130.000	306.9487	0.00	1.00	0.00	0.00
140.000	305.2022	0.00	1.00	0.00	0.00
150.000	303.4607	0.00	1.00	0.00	0.00
160.000	301.7242	0.00	1.00	0.00	0.00
170.000	299.9926	0.00	1.00	0.00	0.00
180.000	298.2661	0.00	1.00	0.00	0.00
190.000	296.5445	0.00	1.00	0.00	0.00
200.000	294.8279	0.00	1.00	0.00	0.00
210.000	293.1162	0.00	1.00	0.00	0.00
220.000	291.4096	0.00	1.00	0.00	0.00
230.000	289.7079	0.00	1.00	0.00	0.00
240.000	288.0113	0.00	1.00	0.00	0.00
250.000	286.3196	0.00	1.00	0.00	0.00
260.000	284.6329	0.00	1.00	0.00	0.00
270.000	282.9511	0.00	1.00	0.00	0.00
280.000	281.2744	0.00	1.00	0.00	0.00
290.000	279.6026	0.00	1.00	0.00	0.00
300.000	277.9358	0.00	1.00	0.00	0.00
350.000	277.9358	0.00	1.00	0.00	0.00
400.000	277.9358	0.00	1.00	0.00	0.00
450.000	277.9358	0.00	1.00	0.00	0.00

500.000	277.9358	0.00	1.00	0.00	0.00
550.000	277.9358	0.00	1.00	0.00	0.00
600.000	277.9358	0.00	1.00	0.00	0.00
650.000	277.9358	0.00	1.00	0.00	0.00
700.000	277.9358	0.00	1.00	0.00	0.00
750.000	277.9358	0.00	1.00	0.00	0.00
800.000	277.9358	0.00	1.00	0.00	0.00
850.000	277.9358	0.00	1.00	0.00	0.00
900.000	277.9358	0.00	1.00	0.00	0.00
950.000	277.9358	0.00	1.00	0.00	0.00
1000.000	277.9358	0.00	1.00	0.00	0.00
1050.000	277.9358	0.00	1.00	0.00	0.00
1100.000	277.9358	0.00	1.00	0.00	0.00
1150.000	277.9358	0.00	1.00	0.00	0.00
1200.000	277.9358	0.00	1.00	0.00	0.00
1250.000	277.9358	0.00	1.00	0.00	0.00
1300.000	277.9358	0.00	1.00	0.00	0.00
1350.000	277.9358	0.00	1.00	0.00	0.00
1400.000	277.9358	0.00	1.00	0.00	0.00
1450.000	277.9358	0.00	1.00	0.00	0.00
1500.000	277.9358	0.00	1.00	0.00	0.00

scaffip.i

```
0              Debug Option
3              Print Option
5.0            Source Height
1.0            Receiver Height
10000.0        Horizontal Range
0.0            Init. Scan Angle
45.0           Final Scan Angle
15.0           Angular Resolution
100.0          Frequency
130.0          Source Level
30.0           Background Level
1              # of Ground Layers
366.           Flow Resistivity
0.27           Omega
0.5            Pore Shape Factor
0.5            Grain Shape Factor
```

Output File

scaffip.o

```
                    Scanning Fast Field Program (SCAFFIP)
                                EOSAEL Version
--------------------------------------------------------------------------------
SOURCE HEIGHT    =  5.0 METERS          FREQUENCY          = 100.0 Hz
DETECTOR HEIGHT  =  1.0 METERS          DETECTOR CENTERED
SOURCE LEVEL     =  130.0 DECIBELS      BACKGROUND LEVEL   = 30.0 DECIBELS
MAXIMUM RANGE    =  10000.0 METERS
--------------------------------------------------------------------------------
              DETECTION RANGE (METERS) OVER BENIGN TERRAIN
--------------------------------------------------------------------------------
               |                  SIGNAL-NOISE (dB)
AZIMUTH (DEG)  |    -10         -5          0           5          10
--------------------------------------------------------------------------------
     0.0           8250.8      6371.5      4948.7      4914.3      3493.7
    15.0           8250.8      6371.5      4948.7      4914.3      3493.7
    30.0           8250.8      6371.5      4948.7      4914.3      3493.7
--------------------------------------------------------------------------------
```

Appendix C
Debug Features

As mentioned in section 3.2, SCAFFIP allows for input of a debug code to allow for modification of some of the default parameters and provide some additional output information. This allows for a standardization of the input and output format, yet allows for versatility. Currently, the debug files consist of up to six files: two input files and four output files.

Input Files

Code: 8

File Name: extra.los

Format:

extra

where

extra - value for extra loss typically between 10^{-6} to 10^{-2} (default 10^{-4})

Code: 16

File Name: trans.frm

Format:

points Npan

where

points	- number of points to use in calculations. Must be power of 2. (default 1024)
Npan	- set the number of panels to use in calculation. Npan < 0 Calculate number of panels (default -2) Npan > 0 Set number of panels

Output Files

Code: 1

File Name: sound.o

Format:

```
  Z(1)     C(1)
  Z(2)     C(2)
   .        .
            .
   .        .
Z(Nint)  C(Nint)
```

where

Nint - number of interfaces.

Z(i) - height of the ith interface in meters.

C(i) - sound speed of the ith interface in m/s.

Code: 32

File Name: npan.e

Format:

```
Kmax     Nyq    Npan
Kwidth   DelR   Delk
```

where

Kmax	-	maximum wave number in summation
DelR	-	range increment
Delk	-	wavenumber increment
Npan	-	number of panels
Kwidth	-	wavenumber band width per panel
Nyq	-	nyquist criteria

Code: 64

File Name: imped.e

Format:

```
Zreal    Zimg
Kreal    Kimg
```

where

Zreal, Zimg	-	real and imaginary part of the ground impedance
Kreal, Kimg	-	real and imaginary part of the wavenumber

Code: 128

File Name: wavnum.e

Format:

```
    wavenum(1)          mag(1)
    wavenum(2)          mag(2)
        .  .               .
        .  .               .
        .  .               .
  wavenum(points)     mag(points)
```

where

points	-	number of points in transform
wavnum(i)	-	wavenumber of ith point
mag(i)	-	magnitude of the kernel of the ith point

Appendix D
User Developed Interface Requirements for SCAFFIP

This appendix discusses the possibility of the user wishing to develop a personalized front end user interface for SCAFFIP. As mentioned in section 1, deviations from the standard user interface supported by ARL/BED will not automatically provide debugging assistance. Future upgrades will conform to the standard interface and will not be integrated into a user developed interface. This policy is in effect because of funding and personnel shortages. Limited assistance can be provided for input data and problems related to the propagation models. Appendix A contains the complete source code listing for the model.

The front end user interface for SCAFFIP is configured so the five input files are read. There are some additional input files that may be read; however, these files deal with the debug number read in. The five primary input files are discussed here. Debug input files are discussed in appendix C. The front-end user interface files for SCAFFIP are main.c, input.c, header.c, parser.c, and sound.c. The file main.c is the main driver program that calls all the other parts of the program. The file input.c is the function that reads the five input files and calls parser to decipher the debug code. The file header.c provides a visual printout of the input parameters. The file sound.c calculates the sound speed profile from the meteorological data and the given geometry. Because the sound speed profile is dependent on a vector relationship between the wind direction and the direction of propagation, the sound speed profile is calculated at each azimuth value. The functions input and sound initialize three of the structures contained in the include file ffp.h.

To develop a new user interface, the new user interface must initialize these three structures: (1) geometry, (2) ground, and (3) interface. The include file ffp.h contains two other structures; however, these structures are initialized and used within propmod exclusively. The geometry structure contains the parameters that describe the structure of the geometry of the problem: source height, receiver height, horizontal range, starting scan angle, ending scan angle, and angular resolution of scan. The ground structure contains all of the parameters for calculating the ground impedance using Attenborough's Four Parameter Model (see section 2.4). The interface structure contains the meteorological profile data. Although this data is used in the function sound, some of this information is required by the function propmod to calculate the molecular absorption coefficients (see section 2.3). While modifying the standard user interface, the user may choose to input the values for the ground impedance directly instead of using the built-in ground impedance model. This can be accomplished by storing the values into the structure gp and replacing the variable ground in the argument list for propmod by a variable of type gp that contains the user input ground impedance value. An alternative method for storing the ground impedance is to use one of the unused debug numbers and place a conditional statement to read the file containing the ground impedance values or use the ground impedance model to calculate the values. The second method is preferred because it conforms better to the programming philosophy used in the code development.

After the three structures are initialized, the control can be passed off to the propagation function propmod. The argument list for propmod is

a	-	interface structure array
fd	-	integer file pointer to a binary output file
fp	-	file pointer to an ASCII output file
frequency	-	frequency of interest
src_level	-	source level
back_level	-	background level
geometry	-	geometry structure
ground	-	ground structure
n_ground_layers	-	number of ground layers
n_interfaces	-	number of atmospheric interfaces
dflags	-	array of debug parameters.

When the function propmod is called, these are the arguments that must be passed. These are the areas that must be considered when developing a new user interface.

If the output file needs to be modified, the computer code for storing the model results is located at the end of the function propmod. The open statement for the output file is located in main because main controls the scanning loop. If the number of files opened at once were to be minimized, the open statement could be placed in propmod; however, the open statement would have to be in setup to always append to the file. This method would only allow the output file to be open when there is data to be written. As an error check, a statement would have to be added to check for the presence of the output file at the beginning of the program to prevent appending to a file already present.

These are the basic areas in which the user may wish to alter the program. Other areas may deal with the input and output of the parameters associated with the debug parameters. These items are discussed in appendix C.

Distribution

	Copies
Commandant U.S. Army Chemical School ATTN: ATZN-CM-CC (Mr. Barnes) Fort McClellan, AL 36205-5020	1
NASA Marshal Space Flight Center Deputy Director Space Science Laboratory Atmospheric Sciences Division ATTN: E501 (Dr. Fichtl) Huntsville, AL 35802	1
NASA/Marshall Space Flight Center Atmospheric Sciences Division ATTN: Code ED-41 Huntsville, AL 35812	1
Deputy Commander U.S. Army Strategic Defense Command ATTN: CSSD-SL-L (Dr. Lilly) P.O. Box 1500 Huntsville, AL 35807-3801	1
Deputy Commander U.S. Army Missile Command ATTN: AMSMI-RD-AC-AD (Dr. Peterson) Redstone Arsenal, AL 35898-5242	1
Commander U.S. Army Missile Command ATTN: AMSMI-RD-DE-SE (Mr. Lill, Jr.) Redstone Arsenal, AL 35898-5245	1

Commander
U.S. Army Missile Command
ATTN: AMSMI-RD-AS-SS (Mr. Anderson) 1
Redstone Arsenal, AL 35898-5253

Commander
U.S. Army Missile Command
ATTN: AMSMI-RD-AS-SS (Mr. B. Williams) 1
Redstone Arsenal, AL 35898-5253

Commander
U.S. Army Missile Command
Redstone Scientific Information Center
ATTN: AMSMI-RD-CS-R/Documents 1
Redstone Arsenal, AL 35898-5241

Commander
U.S. Army Aviation Center
ATTN: ATZQ-D-MA (Mr. Heath) 1
Fort Rucker, AL 36362

Commander
U.S. Army Intelligence Center
and Fort Huachuca
ATTN: ATSI-CDC-C (Mr. Colanto) 1
Fort Huachuca, AZ 85613-7000

Northrup Corporation
Electronics Systems Division 1
ATTN: Dr. Tooley
2301 West 120th Street, Box 5032
Hawthorne, CA 90251-5032

Commander
Pacific Missile Test Center
Geophysics Division
ATTN: Code 3250 (Mr. Battalino) 1
Point Mugu, CA 93042-5000

Commander
Code 3331
Naval Weapons Center
ATTN: Dr. Shlanta 1
China Lake, CA 93555

Lockheed Missiles & Space Co., Inc.
Kenneth R. Hardy
ORG/91-01 B/255 1
3251 Hanover Street
Palo Alto, CA 94304-1191

Commander
Naval Ocean Systems Center
ATTN: Code 54 (Dr. Richter) 1
San Diego, CA 92152-5000

Meteorologist in Charge
Kwajalein Missile Range
P.O. Box 67 1
APO San Francisco, CA 96555

U.S. Department of Commerce Center
Mountain Administration
Support Center, Library, R-51
Technical Reports
325 S. Broadway 1
Boulder, CO 80303

Dr. Hans J. Liebe
NTIA/ITS S 3
325 S. Broadway 1
Boulder, CO 80303

NCAR Library Serials
National Center for Atmos Research
P.O. Box 3000 1
Boulder, CO 80307-3000

Headquarters
Department of the Army
ATTN: DAMI-POI 1
Washington, DC 20310-1067

Mil Asst for Env Sci Ofc of
the Undersecretary of Defense
for Rsch & Engr/R&AT/E&LS
Pentagon - Room 3D129 1
Washington, DC 20301-3080

Headquarters
Department of the Army
DEAN-RMD/Dr. Gomez 1
Washington, DC 20314

Director
Division of Atmospheric Science
National Science Foundation
ATTN: Dr. Bierly 1
1800 G. Street, N.W.
Washington, DC 20550

Commander Space & Naval Warfare System Command ATTN: PMW-145-1G Washington, DC 20362-5100	1
Director Naval Research Laboratory ATTN: Code 4110 (Mr. Ruhnke) Washington, DC 20375-5000	1
Commandant U.S. Army Infantry ATTN: ATSH-CD-CS-OR (Dr. E. Dutoit) Fort Benning, GA 30905-5090	1
USAFETAC/DNE Scott AFB, IL 62225	1
Air Weather Service Technical Library - FL4414 Scott AFB, IL 62225-5458	1
USAFETAC/DNE ATTN: Mr. Glauber Scott AFB, IL 62225-5008	1
Headquarters AWS/DOO Scott AFB, IL 62225-5008	1
Commander U.S. Army Combined Arms Combat ATTN: ATZL-CAW Fort Leavenworth, KS 66027-5300	1

Commander U.S. Army Space Institute ATTN: ATZI-SI Fort Leavenworth, KS 66027-5300	1
Commander U.S. Army Space Institute ATTN: ATZL-SI-D Fort Leavenworth, KS 66027-7300	1
Commander Phillips Lab ATTN: PL/LYP (Mr. Chisholm) Hanscom AFB, MA 01731-5000	1
Director Atmospheric Sciences Division Geophysics Directorate Phillips Lab ATTN: Dr. McClatchey Hanscom AFB, MA 01731-5000	1
Raytheon Company Dr. Sonnenschein Equipment Division 528 Boston Post Road Sudbury, MA 01776 Mail Stop 1K9	1
Director U.S. Army Materiel Systems Analysis Activity ATTN: AMXSY-CR (Mr. Marchetti) Aberdeen Proving Ground, MD 21005-5071	1

Director U.S. Army Materiel Systems Analysis Activity ATTN: AMXSY-MP (Mr. Cohen) Aberdeen Proving Ground, MD 21005-5071	1
Director U.S. Army Materiel Systems Analysis Activity ATTN: AMXSY-AT (Mr. Campbell) Aberdeen Proving Ground, MD 21005-5071	1
Director U.S. Army Materiel Systems Analysis Activity ATTN: AMXSY-CS (Mr. Bradley) Aberdeen Proving Ground, MD 21005-5071	1
Director ARL Chemical Biology Nuclear Effects Division ATTN: AMSRL-SL-CO Aberdeen Proving Ground, MD 21010-5423	1
Army Research Laboratory ATTN: AMSRL-D 2800 Powder Mill Road Adelphi, MD 20783-1145	1
Army Research Laboratory ATTN: AMSRL-OP-SD-TP Technical Publishing 2800 Powder Mill Road Adelphi, MD 20783-1145	1

Army Research Laboratory
ATTN: AMSRL-OP-CI-SD-TL 1
2800 Powder Mill Road
Adelphi, MD 20783-1145

Army Research laboratory
ATTN: AMSRL-SS-SH 1
(Dr. Sztankay)
2800 Powder Mill Road
Adelphi, MD 20783-1145

U.S. Army Space Technology
and Research Office
ATTN: Ms. Brathwaite 1
5321 Riggs Road
Gaithersburg, MD 20882

National Security Agency
ATTN: W21 (Dr. Longbothum) 1
9800 Savage Road
Fort George G. Meade, MD 20755-6000

OIC-NAVSWC
Technical Library (Code E-232) 1
Silver Springs, MD 20903-5000

Commander
U.S. Army Research office
ATTN: DRXRO-GS (Dr. Flood) 1
P.O. Box 12211
Research Triangle Park, NC 27009

Dr. Jerry Davis
North Carolina State University
Department of Marine, Earth, and
Atmospheric Sciences 1
P.O. Box 8208
Raleigh, NC 27650-8208

Commander
U.S. Army CECRL
ATTN: CECRL-RG (Dr. Boyne) 1
Hanover, NH 03755-1290

Commanding Officer
U.S. Army ARDEC
ATTN: SMCAR-IMI-I, Bldg 59 1
Dover, NJ 07806-5000

Commander
U.S. Army Satellite Comm Agency
ATTN: DRCPM-SC-3 1
Fort Monmouth, NJ 07703-5303

Commander
U.S. Army Communications-Electronics
Center for EW/RSTA
ATTN: AMSEL-EW-MD 1
Fort Monmouth, NJ 07703-5303

Commander
U.S. Army Communications-Electronics
Center for EW/RSTA
ATTN: AMSEL-EW-D 1
Fort Monmouth, NJ 07703-5303

Commander U.S. Army Communications-Electronics Center for EW/RSTA ATTN: AMSEL-RD-EW-SP Fort Monmouth, NJ 07703-5206	1
Commander Department of the Air Force OL/A 2d Weather Squadron (MAC) Holloman AFB, NM 88330-5000	1
PL/WE Kirtland AFB, NM 87118-6008	1
Director U.S. Army TRADOC Analysis Center ATTN: ATRC-WSS-R White Sands Missile Range, NM 88002-5502	1
Director U.S. Army White Sands Missile Range Technical Library Branch ATTN: STEWS-IM-IT White Sands Missile Range, NM 88002	3
Army Research Laboratory ATTN: AMSRL-BE (Mr. Veazy) Battlefield Environment Directorate White Sands Missile Range, NM 88002-5501	1
Army Research Laboratory ATTN: AMSRL-BE-A (Mr. Rubio) Battlefield Environment Directorate White Sands Missile Range, NM 88002-5501	1

Army Research Laboratory ATTN: AMSRL-BE-M (Dr. Niles) Battlefield Environment Directorate White Sands Missile Range, NM 88002-5501	1
Army Research Laboratory ATTN: AMSRL-BE-W (Dr. Seagraves) Battlefield Environment Directorate White Sands Missile Range, NM 88002-5501	1
USAF Rome Laboratory Technical Library, FL2810 Corridor W, STE 262, RL/SUL 26 Electronics Parkway, Bldg 106 Griffiss AFB, NY 13441-4514	1
AFMC/DOW Wright-Patterson AFB, OH 03340-5000	1
Commandant U.S. Army Field Artillery School ATTN: ATSF-TSM-TA (Mr. Taylor) Fort Sill, OK 73503-5600	1
Commander U.S. Army Field Artillery School ATTN: ATSF-F-FD (Mr. Gullion) Fort Sill, OK 73503-5600	1
Commander Naval Air Development Center ATTN: Al Salik (Code 5012) Warminister, PA 18974	1

Commander U.S. Army Dugway Proving Ground ATTN: STEDP-MT-M (Mr. Bowers) Dugway, UT 84022-5000	1
Commander U.S. Army Dugway Proving Ground ATTN: STEDP-MT-DA-L Dugway, UT 84022-5000	1
Defense Technical Information Center ATTN: DTIC-OCP Cameron Station Alexandria, VA 22314-6145	2
Commander U.S. Army OEC ATTN: CSTE-EFS Park Center IV 4501 Ford Ave Alexandria, VA 22302-1458	1
Commanding Officer U.S. Army Foreign Science & Technology Center ATTN: CM 220 7th Street, NE Charlottesville, VA 22901-5396	1
Naval Surface Weapons Center Code G63 Dahlgren, VA 22448-5000	1

Commander and Director U.S. Army Corps of Engineers Engineer Topographics Laboratory ATTN: ETL-GS-LB Fort Belvoir, VA 22060	1
U.S. Army Topo Engineering Center ATTN: CETEC-ZC Fort Belvoir, VA 22060-5546	1
Commander USATRADOC ATTN: ATCD-FA Fort Monroe, VA 23651-5170	1
TAC/DOWP Langley AFB, VA 23665-5524	1
Commander Logistics Center ATTN: ATCL-CE Fort Lee, VA 23801-6000	1
Science and Technology 101 Research Drive Hampton, VA 23666-1340	1
Commander U.S. Army Nuclear and Chemical Agency ATTN: MONA-ZB, Bldg 2073 Springfield, VA 22150-3198	1
Record Copy	3
Total	89